ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN GÉNIE
CONCENTRATION RÉSEAUX DE TÉLÉCOMMUNICATIONS
M. Ing.

PAR
François GOURLAY

CARACTÉRISATION EN PUISSANCE DES INSTRUCTIONS D'UN PROCESSEUR MULTICOEUR ASYNCHRONE

MONTRÉAL, LE 12 JUILLET 2012

PRÉSENTATION DU JURY

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

M. François Gagnon, directeur de mémoire
Département de Génie électrique à l'École de technologie supérieure

M. Claude Thibeault, codirecteur de mémoire
Département de Génie électrique à l'École de technologie supérieure

M. Nicolas Constantin, président du jury
Département de Génie électrique à l'École de technologie supérieure

M. Yvon Savaria, membre externe du jury
Département de Génie électrique à l'École Polytechnique de Montréal

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 26 JUIN 2012

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

REMERCIEMENTS

Je remercie le Professeur François Gagnon, mon directeur, pour sa très grande disponibilité et sa clairvoyance à me diriger.

Je remercie également le Professeur Claude Thibeault, mon co-directeur, pour l'infinie patience dont il a fait preuve à me guider sur les chemins de la rigueur.

CARACTÉRISATION EN PUISSANCE DES INSTRUCTIONS D'UN PROCESSEUR MULTICOEUR ASYNCHRONE

François GOURLAY

RÉSUMÉ

Ce travail porte sur une technique d'estimation de consommation de puissance et d'énergie pour des programmes logiciels exécutés par un processeur multicoeur asynchrone en traitement parallèle. Cette technique se veut d'abord et avant tout simple et de première estimation. L'application de cette technique permet dans un premier temps de caractériser chaque instruction en assembleur en termes de fréquence d'exécution propre et de temps d'exécution distinct. Ces temps d'exécution sont ensuite utilisés dans le calcul théorique d'un modèle de consommation de puissance et d'énergie représentant un programme logiciel exécutant ces mêmes instructions. Les estimations théoriques des consommations de puissance et d'énergie sont validées en les comparant aux mesures expérimentales de consommation effectuées. Des erreurs relatives de moins de 1% sont obtenues pour des programmes simples et des erreurs relatives variant de 6.1% à 9.6% pour les programmes plus complexes. Cette technique permet de connaître très tôt dans le processus de conception la consommation énergétique d'un programme logiciel spécifique exécuté avec un processeur asynchrone. Ultimement, cette technique simple pourrait aider à réduire la consommation énergétique d'un processeur asynchrone en guidant le concepteur dans ses choix algorithmiques.

Mots clés: consommation puissance VLSI

CARACTÉRISATION EN PUISSANCE DES INSTRUCTIONS D'UN PROCESSEUR MULTICOEUR ASYNCHRONE

François GOURLAY

ABSTRACT

This thesis proposes a technique to estimate power and energy consumption of software programs executed by an asynchronous multicore processor using parallel processing. This technique is simple and can be used to produce early estimates. It allows the characterization of each assembler instruction in terms of its own frequency and its distinct execution time. This execution time parameter is then used in the theoretical calculations of a power and energy consumption model representing the software executing those same instructions. The theoretical power and energy consumption calculations are then validated with their experimental counterparts. Relative errors of less than 1% for simple programs and relative errors varying from 6.1% to 9.6% for more complex programs are achieved. This simple technique could be used to find, very early in the design process, the energy consumption of a specific software program using an asynchronous multicore processor. Ultimately, this simple technique should reduce the power consumption of an asynchronous multicore processor in guiding the designer in his/her choices of algorithms.

Keywords: power consumption VLSI

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

ADD	Addition
ALU	Arithmetic logic unit
DSP	Digital signal processor
FIR	Finite impulse response
MAC	Multiply and accumulate
OPÉRA	Octasic Polytechnique ÉTS Radio Application
TP	Test point
VEB	Vocallo evaluation board

INTRODUCTION

La motivation première de ce travail est issue d'un constat relativement récent et remarquable: «As transistor counts and clock frequencies have increased, power consumption has skyrocketed and now is a primary design constraint » Weste et Harris (2005, p. 186) [32]. C'est au début des années 1990 que la consommation de puissance des puces à haute densité en transistors, « very large scale integration system», VLSI, attire l'attention des chercheurs.

En 1993, Hernandez *et al.* [9] ainsi que Callaway et Swartzlander [4], sont les premiers à proposer un modèle d'estimation de la consommation de puissance des puces VLSI au niveau des circuits et des portes logiques. Au cours des deux décennies suivantes, une multitude de chercheurs apporteront leurs contributions à ce domaine.

Ces dernières années, 2010 et 2011, verront apparaître des méthodologies complètes d'estimation de consommation de puissance directement orientées vers la conception complète des processeurs ou de circuits VLSI. C'est à Jevtic et Carreras [14] ainsi que Rethinagiri *et al.* [19] que l'on doit la création de telles méthodologies de conception.

Malgré tous ces apports technologiques, avec la miniaturisation incessante des circuits intégrés, des applications de plus en plus performantes et énergivores apparaissent chaque jour, et cela ne semble pas vouloir s'arrêter, bien au contraire! En effet, ces centaines de millions de transistors miniaturisés sur des surfaces de plus en plus petites, demandent toujours plus d'énergie pour fonctionner. Malheureusement, les fabricants de piles n'arrivent pas à suivre le rythme de croissance en demande énergétique.

Une des pistes de solutions afin d'économiser le plus possible cette précieuse énergie, est de concevoir certaines applications, comme par exemple les filtres numériques, avec les instructions les moins énergivores possibles. Ultimement, le design même des processeurs utilisant un choix d'algorithmes moins énergivores est à considérer.

Par contre, toutes ces avancées technologiques ciblaient généralement les circuits synchrones et plus particulièrement les processeurs synchrones, laissant pour compte les processeurs asynchrones dans un quasi oubli. Ce mémoire s'inscrit dans la foulée du projet OPÉRA en présentant la caractérisation en puissance et en énergie, d'instructions du processeur multicoeur asynchrone de la carte Vocallo d'Octasic.

> «L'objectif global du projet OPÉRA est d'explorer différentes classes d'applications par rapport à une implémentation matérielle existante, de manière à guider les phases de configuration, de vérification et possiblement de modification du design actuel. De manière plus spécifique, le projet de recherche OPÉRA vise à déterminer si la classe d'algorithmes choisie peut en effet être implémentée de manière efficace et correcte sur l'architecture étudiée.» (Tremblay, 2009, p. 6 [29])

Ce mémoire présente d'abord un aperçu de l'état de l'art et positionne les présents travaux. Avec le second chapitre, l'environnement de tests utilisé, tant matériel que logiciel, est présenté. Les chapitres suivants se concentrent sur la paramétrisation, la caractérisation et enfin la modélisation des instructions nécessaires au bon fonctionnement d'un filtre à réponse impulsionnelle finie (Finite Impulse Response, « FIR ») conventionnel. Les chapitres subséquents présentent l'application d'un modèle mathématique énergétique d'un filtre réel. Il est possible de prédire avec une précision raisonnable la consommation énergétique d'un filtre « FIR » typique. Le dernier chapitre explore l'utilisation des produits d'intermodulation, générés par les cœurs du processeur asynchrone étudié, dans les processus de caractérisation.

Dans l'ensemble, les présents travaux proposent une technique distincte d'estimation de consommation de puissance et d'énergie pour des programmes logiciels codés en assembleur et exécutés par un processeur multicoeur asynchrone en traitement parallèle. Cette technique se veut simple et de première approximation. Cette technique, de même que les efforts de caractérisation poussés du processeur multicoeur asynchrone, constituent les principales contributions de ce mémoire.

CHAPITRE 1

L'ÉTAT DE L'ART

1.1 Revue des connaissances

En 1993, Hernandez *et al.* [9] ainsi que Callaway et Swartzlander [4], sont les premiers à proposer des modèles d'estimation de la consommation de puissance des puces VLSI au niveau des circuits et des portes logiques. Pour la première fois, en 1995, Tiwari *et al.* [25] ainsi que Mike *et al.* [17] proposent des modèles basés sur les instructions utilisées d'un programme.

En 1996, Wang et Roy [30] se concentrent sur la consommation maximale instantanée de puissance dissipée dans les puces VLSI. Leurs travaux guideront la conception future des circuits VLSI. Durant la même période, Ishiara et Yasuura [11] [12], évalueront les diverses méthodes d'estimation de consommation de puissance proposées jusqu'à ce jour.

L'année 1998 voit Katkoori et Vemuri [16] prendre en considération l'analyse de consommation de puissance dans la conception de circuit VLSI. Wang et Roy [31] ainsi que Russell et Jacome [21] proposent des logiciels de tests automatisés complétant la simulation afin d'en déterminer la consommation de puissance.

En 1998, l'apparition d'algorithmes de conception de circuits VLSI touchant la réduction de consommation d'énergie nous est présentée par Nguyen, Chatterjee et Roy [18]. En 1999, Chen, Irwin et Bajwa [5] nous présentent de nouvelles techniques d'estimation de consommation de puissance basées sur la non connaissance des structures internes des circuits logiques. Les années 2000 et 2001 voient les diverses techniques d'estimation de consommation de puissance atteindre un niveau avancé de développement ; que ce soit avec les processeurs superscalaires, Conte *et al.* [6], au niveau de l'ensemble de circuits VLSI, Hsiao, Rudnick et Patel [10] ou encore au niveau des réseaux logiques avec Theodoridis *et al.* [24].

En 2001, Ascia *et al.* [2] étudieront l'impact du logiciel sur la consommation de puissance d'un circuit. Rita, Irwin et Bajwa [20] amélioreront les méthodes d'estimation de la consommation énergétique avant et pendant la conception des circuits. L'année 2002 verra Guitton-Ouhamou, Belleudy et Auguin [8] développer une bibliothèque de l'impact énergétique des différentes instructions en langage d'assembleur d'un processeur. C'est à Tran, Kim et Kim [28] en 2005 et Shin et Lee [22] en 2006 que l'on doit l'apparition de méthodes avancées d'estimation énergétique qui divise le processeur en sections distinctes dont chacune est étudiée séparément et dont la somme énergétique donne la consommation totale du processeur.

Les années 2007 à 2010 verront le développement de plusieurs systèmes d'estimation énergétique prenant en compte le système complet avec son logiciel, sans toutefois connaître la composition intrinsèque des circuits. Plusieurs chercheurs participent à cette évolution. Kang *et al.* [15] en 2007, Jevtic et Carreras [13] ainsi que Cai *et al.* [3] en 2010 et Gonzalez *et al.* [7] en 2011.

De plus, les années 2010 et 2011 verront apparaître des méthodologies complètes d'estimation de consommation de puissance directement orientées vers la conception complète des processeurs ou des circuits VLSI. C'est à Jevtic et Carreras [14] ainsi que Rethinagiri *et al.* [19] que l'on doit la création de telles méthodologies de conception.

En parallèle avec tous ces travaux sur les processeurs synchrones essayant d'éliminer les problèmes de distribution d'horloge, de variation de paramètres, de consommation de puissance et de complexité de conception, quelques groupes de chercheurs se sont penchés sur la conception des processeurs asynchrones visant à éliminer ces problèmes. C'est depuis le milieu des années 80 que les microprocesseurs asynchrones ont vu le jour. Rappelons qu'un processeur asynchrone n'utilise pas d'horloge centrale pour exécuter ses instructions. Il utilise plutôt un protocole simple de signaux indiquant l'achèvement des instructions.

Les méthodologies de conception de ces processeurs ont été abondamment documentées. Pensons aux travaux riches en innovations et abondamment cités de Bainbridge *et al.* [33] utilisant une approche asynchrone basée sur les délais critiques afin de satisfaire les longs chemins d'interconnections. Woods *et al.* [38] réalisent un microprocesseur asynchrone utilisant une approche basée sur le micro-pipeline de Sutherland [39]. Ces travaux sont repris par Furber *et al.* [34] et [35] qui proposent dans un premier temps un micro-pipeline à quatre phases visant à réduire les coûts fixes et dans un deuxième temps, ils conçoivent une puce comprenant un contrôleur asynchrone de 32 bits complet très efficace. Martin *et al.* [36] présente le design asynchrone d'un clone du processeur synchrone MIPS R3000. Stevens *et al.* [37] font une étude décrivant les avantages et les pièges des designs asynchrones appliqués aux architectures de pointe des microprocesseurs. .

Bien que tous ces travaux contournent d'une certaine manière les problèmes de consommation de puissance des processeurs synchrones, aucune technique de mesure de cette même consommation énergétique des instructions de processeurs asynchrones n'a été développée à ce jour. Aucun des efforts de recherche mentionné jusqu'ici n'a porté sur l'étude de la consommation énergétique des processeurs asynchrones.

Les présents travaux proposent de faire la caractérisation en puissance et en énergie, des instructions d'un processeur multicoeur asynchrone.

CHAPITRE 2

ENVIRONNEMENT DE TEST

2.1 Introduction

Ce chapitre présente l'environnement de tests utilisé dans le cadre de ce projet (figure 2.1), pour la caractérisation du processeur multicoeur asynchrone de la carte Vocallo. Cet environnement contient une partie matérielle et une partie logicielle. La partie matérielle comprend la carte Vocallo contenant le processeur « DSP » multicoeur. Cette carte est reliée à un amplificateur, qui lui-même est relié à un analyseur de spectres. La partie logicielle permet de décharger le code étudié directement dans le processeur. Les sections suivantes décrivent en détails chacune de ces parties.

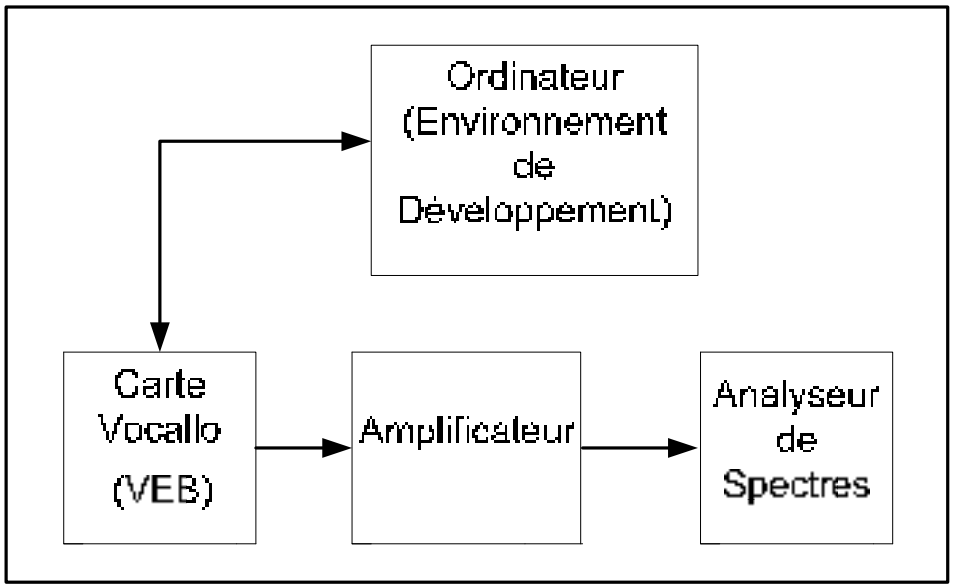


Figure 2.1 Diagramme bloc de l'environnement de test

2.2 Carte Vocallo

Tous les travaux ont été effectués avec la carte d'Octasic, le « Vocallo Evaluation Board OCTVOC-EB Rev.1 » (VEB), présenté à la figure 2.2. Cette carte contient un processeur « DSP » de quinze cœurs intégrés, dans la puce OCT1010, où chacun des cœurs contient également seize « ALU ». Chaque cœur fonctionne en parallèle de façon asynchrone et indépendante, alors que les opérations des « ALU » sont lancées de manière séquentielle dans leur exécution parallèle. J.-P. Tremblay [29] fait une excellente description de l'architecture du « DSP » multicoeur d'Octasic dans son chapitre 3.

La particularité de cette carte est qu'elle contient deux points de tests, TP1 et TP2, reliés par une résistance de précision de 0.005Ω. Au travers de cette résistance passe le courant total circulant dans l'ensemble des quinze cœurs durant l'exécution des programmes. Connaissant la tension d'alimentation des cœurs, qui est mesurable, il devient possible de calculer la puissance de traitement moyenne de chaque « ALU » pour un type spécifique d'instructions, à condition d'amplifier suffisamment la différence de potentiel aux bornes de la résistance de précision afin de mesurer le courant concerné.

Figure 2.2 Carte Vocallo

La figure 2.2 présente le « Vocallo Evaluation Board OCTVOC-EB Rev.1 » (VEB) en mode autonome, alimenté par une source externe. Les pinces alligators jaunes et vertes sont directement branchées aux points de test TP1 et TP2 qui sont, à leur tour, reliées à l'amplificateur.

Les spécifications du manufacturier indiquent le courant maximum total traversant l'ensemble des quinze cœurs est de 1.4 A. Passant par la résistance de 0.005Ω aux points de test TP1 et TP2, ce courant produit une différence de potentiel de 7 mV pour l'ensemble des quinze cœurs soit 467 µV par cœur actif.

2.3 Amplificateur

Le rôle de l'amplificateur, développé dans ce projet, est de ramener la faible différence de potentiel aux bornes de la résistance à un niveau variant de 0 à 5 V. Pour ce faire, le gain de l'amplificateur a été fixé à 8,670. C'est un amplificateur de type différentiel. Il est important de spécifier que compte tenu des spécifications du manufacturier et du produit gain, bande passante de l'amplificateur opérationnel utilisé, les niveaux exacts de puissance des différentes fréquences des captures d'écran de l'analyseur de spectres, ne sont pas pertinents.

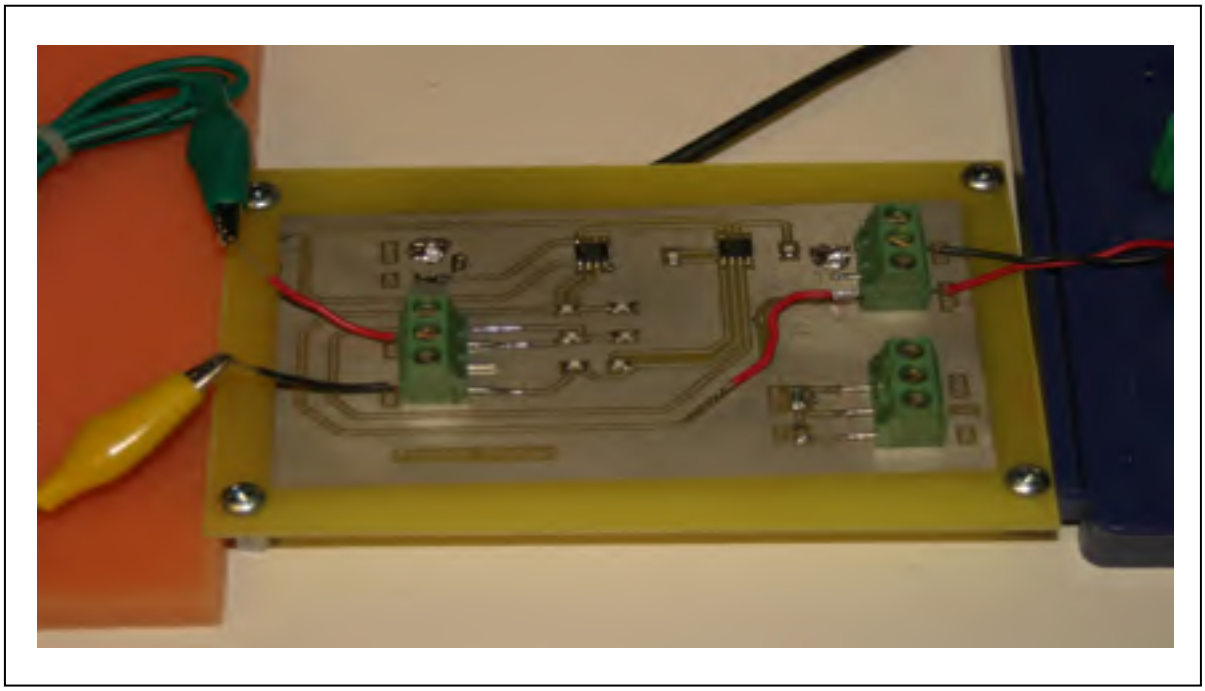


Figure 2.3 Amplificateur

La figure 2.3 montre une photo de l'amplificateur. La photo montre à gauche les pinces alligators branchées des points de test TP1 et TP2 du VEB à l'entrée de l'amplificateur. À droite, la sortie de l'amplificateur est reliée à un analyseur de spectres. L'annexe III nous donne le schéma détaillé de l'amplificateur.

2.4 Environnement physique de test

La figure 2.4 ci-dessous donne un aperçu de l'environnement de test complet. La photo montre à gauche, la carte VEB relié à l'amplificateur, qui à son tour est relié à l'analyseur de spectres. L'ordinateur portatif est branché directement à la carte VEB via un lien Ethernet. Ceci permet de charger directement les programmes de l'environnement de développement à la carte VEB.

Figure 2.4 Environnement de test complet

2.5 Environnement de développement logiciel

L'environnement logiciel de test est fourni par la compagnie Octasic. Cet environnement est du type Microsoft Visual Studio 2008. C'est un environnement propriétaire et personnalisé par la compagnie. La figure 2.5 donne un aperçu de ce type d'environnement de développement.

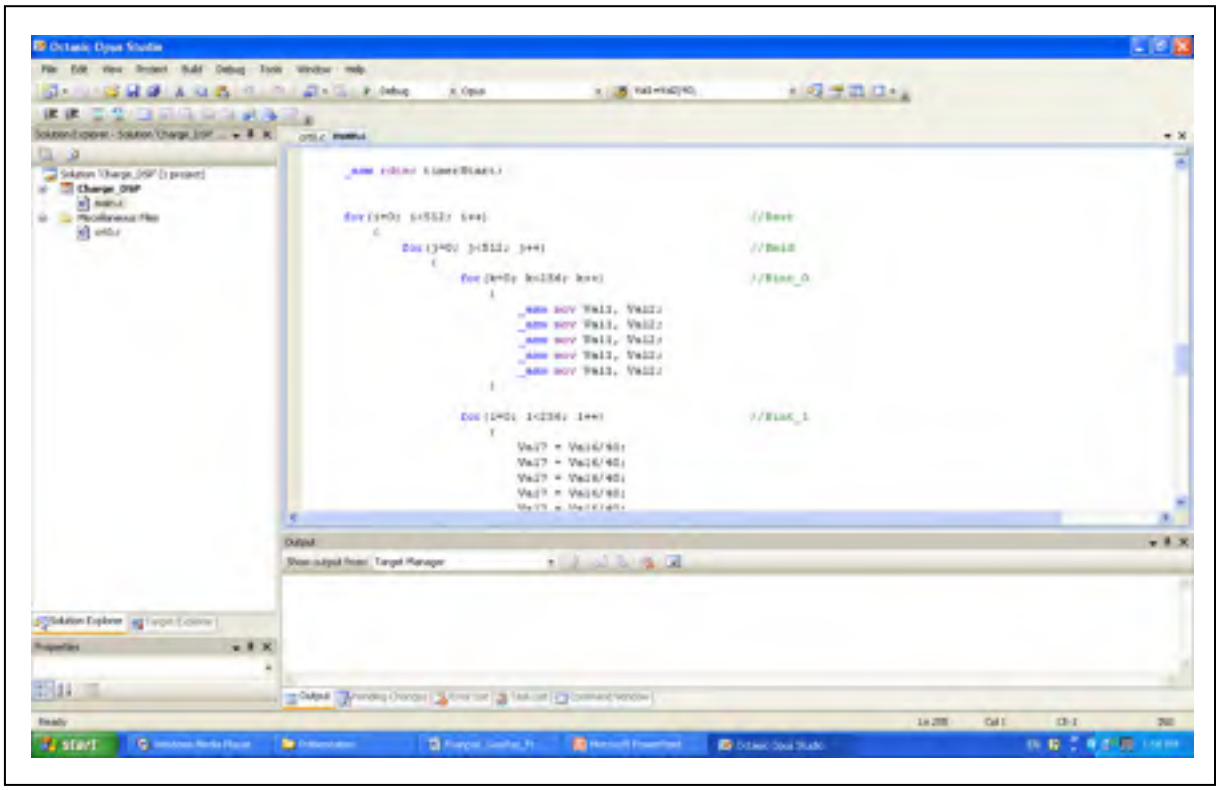


Figure 2.5 Environnement de développement logiciel

2.6 Conclusion

Ce chapitre présente l'environnement de test utilisé dans le cadre de ce projet pour la caractérisation du processeur asynchrone multicoeur, y compris la carte VEB sur laquelle les travaux sont effectués. Compte tenu des faibles tensions de sortie aux points de test, la carte VEB est reliée à un amplificateur, qui lui-même est branché à un analyseur de spectres. Un ordinateur portatif relie directement la carte VEB à l'environnement de développement logiciel.

CHAPITRE 3

PARAMÉTRISATION D'UN PROGRAMME À DOUBLE BOUCLES IMBRIQUÉES

3.1 Introduction

Ce chapitre montre qu'il est possible de paramétrer le temps d'exécution des instructions contenues à l'intérieur de 2 boucles imbriquées, ainsi que le temps d'exécution des boucles elles-mêmes. Les instructions et les boucles paramétrées serviront à définir un modèle mathématique dans un chapitre subséquent.

3.2 Méthodologie

La méthodologie consiste à exécuter un programme en boucle utilisant la même instruction assez longtemps pour atteindre une linéarité du temps d'exécution des cœurs. La durée de l'exécution du programme est mesurée. Cette mesure est effectuée à l'aide du compteur cadencé par une horloge de 25 MHz. Le compteur est activé au début du programme, à l'aide de l'instruction « rdtsc », et désactivé ensuite avec la même instruction à la fin du code étudié. Le compteur donne le nombre de cycles de l'horloge complétés durant la partie étudiée du programme. En multipliant ce nombre de cycles par la période de l'horloge, le temps total d'exécution du code étudié est obtenu. Les résultats du cœur #1 du processeur asynchrone multicoeur sont présentés.

Soient les définitions suivantes :

- $T_{exéc}$: temps d'exécution total mesuré du programme (seconde)
- B_{ext} : nombre d'itérations de la boucle externe (i)
- B_{int} : nombre d'itérations de la boucle interne (j)
- N_{mac} : nombre d'instructions « MAC » contenues dans la boucle interne

```
_asm rdtsc timerStart;

register unsigned int Val1 = 1;
register unsigned int Val2 = 1;
register unsigned int Val3 = 0;

for(i=0; i<600; i++)
    {
        for(j=0; j<65536; j++)
          {
             _asm mac.ll Val3, Val1, Val2;
          }
    }

_asm
    {
        rdtsc timerStop;
        sub timerDiff, timerStart, timerStop;
    }
```

Figure 3.1 Exemple de programme à double boucles imbriquées

La figure 3.1 présente le programme utilisé pour la paramétrisation. L'instruction « MAC » est utilisée à l'intérieur des 2 boucles imbriquées. Ce programme permet la répétition de la même instruction un très grand nombre de fois. Afin d'éliminer la dépendance entre les variables, elles sont initialisées avec des valeurs fixes.

3.3 Paramétrisation

La paramétrisation consiste dans un premier temps à ne faire varier qu'une seule variable à la fois, à savoir le nombre d'instructions « MAC », alors que toutes les autres variables du système étudié, i.e. le nombre de boucles internes et externes, demeurent constantes. Comme le programme étudié consiste en deux boucles imbriquées contenant une instruction spécifique, la paramétrisation, tour à tour, de la boucle interne et de la boucle externe est par la suite effectuée.

3.3.1 Paramétrisation des instructions

Tableau 3.1 Paramétrisation de l'instruction « MAC »

Item	N_{mac}	B_{ext}	B_{int}	Compteur (# cycles)	$T_{exéc}$ (secondes)	Erreur Standard
1	100	1200	65536	614297208	24.57	
2	50	1200	65536	307432055	12.3	
3	25	1200	65536	154010002	6.16	0.0026726
4	10	1200	65536	61858319	2.47	0.0050033
5	5	1200	65536	31163093	1.25	0.0043321
6	0	1200	65536	12945560	0.52	0.2082120

Le tableau 3.1 montre les résultats obtenus pour la paramétrisation de l'instruction « MAC ». Ces résultats sont présentés sous forme graphique à la figure 3.2.

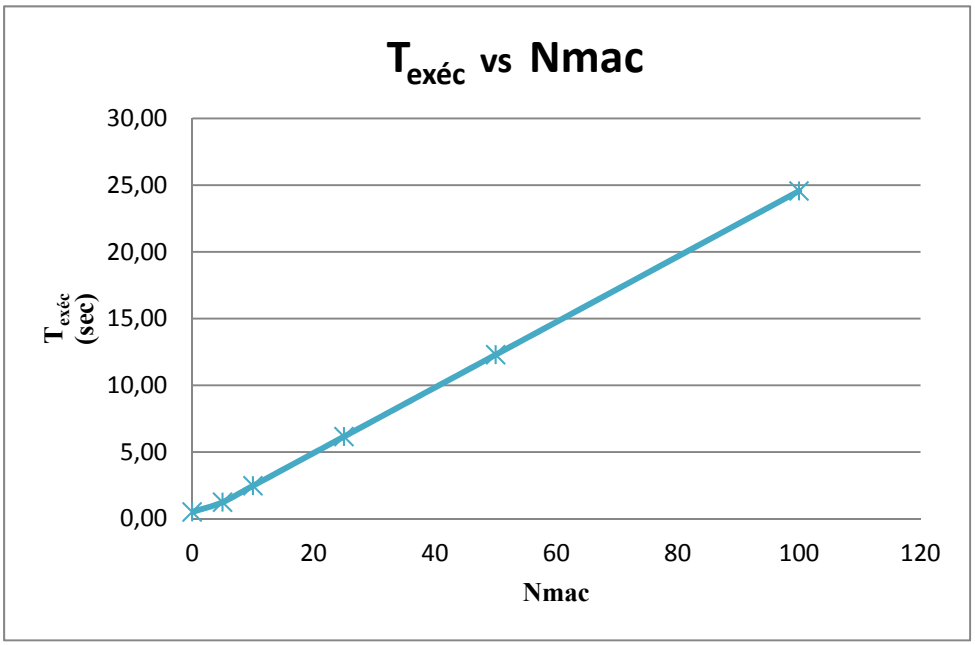


Figure 3.2 Temps d'exécution en fonction du nombre d'instructions « MAC »

La figure 3.2 montre clairement que le temps d'exécution du programme, $T_{exéc}$, par rapport au nombre d'instructions « MAC » utilisé, N_{mac}, est linéaire pour $N_{mac} > 5$. Dans la colonne **Erreur Standard**, du tableau 3.1, la fonction STEYX du logiciel Excel est utilisée afin de déterminer les limites de linéarité de la courbe du graphique de la figure 3.2. La fonction STEYX calcule l'erreur standard de la valeur prédite de $T_{exéc}$, pour chaque valeur de Nmac de la droite de régression passant par ces points. Au tableau 3.1, dans la colonne **Erreur Standard**, à l'item 3, l'erreur standard est calculée en prenant les valeurs de l'item 1 à 3 pour les coordonnées en y de $T_{exéc}$, et les coordonnées correspondantes en x pour Nmac. L'erreur standard de l'item 4 est calculé avec les valeurs $T_{exéc}$ de l'item 1 à 4 et ainsi de suite. La fonction STEYX est également utilisée dans les tableaux 3.2 et 3.3.

3.3.2 Paramétrisation de la boucle interne

Afin de paramétrer la boucle interne du programme étudié, le nombre d'instructions « MAC » utilisé ainsi que le nombre d'itérations de la boucle externe sont gardés constants. Seulement le nombre d'itérations de la boucle interne variera.

Tableau 3.2 Paramétrisation de la boucle interne

Item	**N_{mac}**	**B_{ext}**	**B_{int}**	**Compteur (# cycles)**	**$T_{exéc}$ (secondes)**	**Erreur Standard**
1	50	1200	65536	307515927	12.300637	
2	50	1200	32768	153795062	6.151802	
3	50	1200	16384	76914397	3.076576	0.0004322
4	50	1200	8192	38478075	1.539123	0.0003534
5	50	1200	4096	19238170	0.769527	0.0005735
6	50	1200	2048	9627683	0.385107	0.0005850
7	50	1200	1024	4821530	0.192861	0.0005704
8	50	1200	512	2419896	0.096796	0.0005425
9	50	1200	256	1215252	0.04861	0.0005281
10	50	1200	128	620870	0.024835	0.0004963
11	50	1200	64	316715	0.012669	0.0004760
12	50	1200	32	159163	0.006367	0.0004795
13	50	1200	16	81269	0.003251	0.0004877
14	50	1200	8	42169	0.001687	0.0004950
15	50	1200	4	22467	0.000899	0.0004999
16	50	1200	2	12661	0.000506	0.0005019
17	50	1200	1	7839	0.000314	0.0005013
18	50	1200	0	2046	0.000082	0.0005013

Le tableau 3.2 montre les résultats de la paramétrisation de la boucle interne du programme. Ces résultats sont présentés sous forme graphique à la figure 3.3.

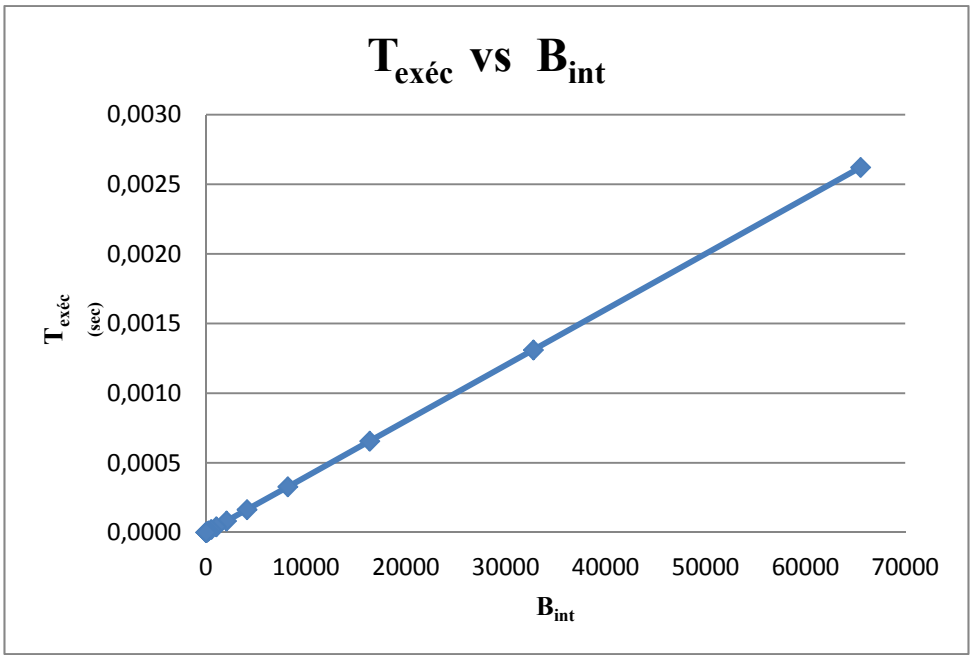


Figure 3.3 Temps d'exécution en fonction du nombre d'itérations de la boucle interne

La figure 3.3 montre clairement que le temps d'exécution du programme, $T_{exéc}$, en fonction du nombre d'itérations de la boucle interne, B_{int}, est linéaire, si on se fie à l'erreur standard.

3.3.3 Paramétrisation de la boucle externe

Afin de paramétrer la boucle externe du programme, le nombre d'instructions « MAC » utilisé est gardé constant ainsi que le nombre d'itérations de la boucle interne. Seulement le nombre d'itérations de la boucle externe varie.

Tableau 3.3 Paramétrisation de la boucle externe

Item	N_{mac}	B_{ext}	B_{int}	Compteur (# cycles)	$T_{exéc}$ (secondes)	Erreur Standard
1	50	8192	65536	2.10E+09	83.94586	
2	50	4096	65536	1.05E+09	41.97407	
3	50	2048	65536	524772471	20.99090	0.0014550
4	50	1024	65536	262408640	10.49635	0.0011914
5	50	512	65536	131217817	5.24871	0.0012968
6	50	256	65536	65597260	2.62389	0.0016225
7	50	128	65536	32803438	1.31214	0.0016671
8	50	64	65536	16423835	0.65695	0.0015631
9	50	32	65536	8216109	0.32864	0.0015039
10	50	16	65536	4111032	0.16444	0.0014639
11	50	8	65536	2060900	0.08244	0.0014214
12	50	4	65536	1033308	0.04133	0.0013856
13	50	2	65536	520261	0.02081	0.0013504
14	50	1	65536	265324	0.01061	0.0013129
15	50	0	65536	2	8.00E-08	0.0012959

Le tableau 3.3 montre les résultats de la paramétrisation de la boucle externe du programme. Ces résultats sont présentés sous forme graphique à la figure 3.4.

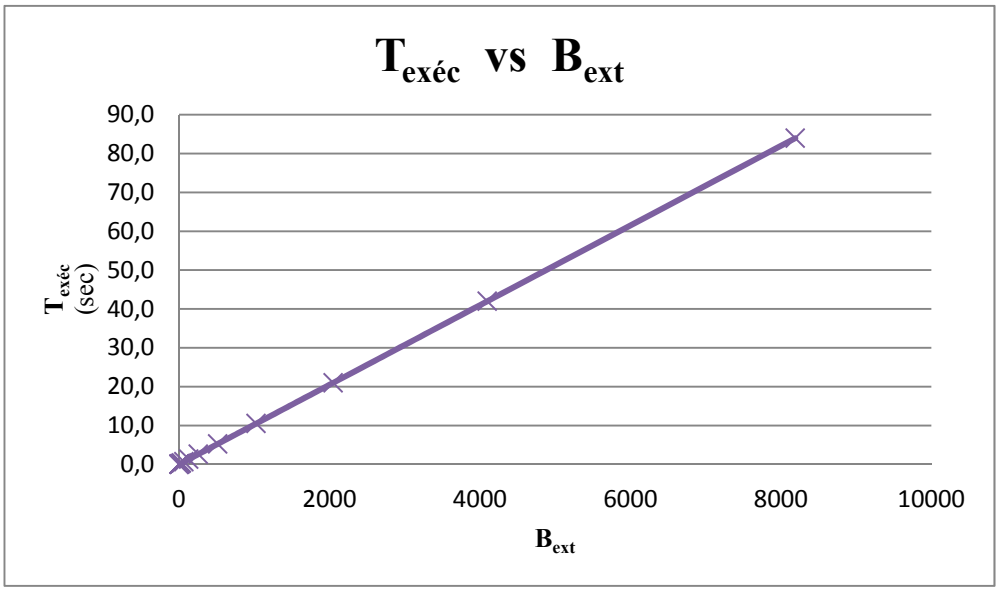


Figure 3.4 Temps d'exécution en fonction du nombre d'itérations de la boucle externe

La figure 3.4 montre clairement que le temps d'exécution du programme, $T_{exéc}$, par rapport au nombre d'itérations de la boucle externe, B_{ext}, est linéaire.

3.4 Conclusion

Dans ce chapitre, un programme simple à 2 boucles imbriquées a été paramétré. Ceci impliquait la paramétrisation de la boucle externe, de la boucle interne et des instructions contenu dans la boucle interne du programme. Dans tous les cas il a été montré que les relations sont linéaires à partir de certaines valeurs. Ceci suggère que les coûts fixes en termes de temps d'exécution deviennent alors négligeables. Les coûts fixes sont définis comme étant la partie non linéaire de la courbe, qui se situe au tout début de celle-ci.

CHAPITRE 4

MODÉLISATION DU TEMPS D'EXÉCUTION DES INSTRUCTIONS

4.1 Introduction

Dans le chapitre précédent, le programme de 2 boucles imbriquées contenant l'instruction « MAC » a été paramétré. Il a été montré que chacune des paramétrisations des boucles interne et externe ainsi que de l'instruction « MAC » devient linéaire après un certain temps d'exécution. Ce chapitre convertira en modèle mathématique chacune de ces paramétrisations.

4.2 Méthodologie

La méthodologie utilisée consiste à poser en hypothèse des équations linéaires de modélisation afin de calculer théoriquement les temps d'exécution des différents éléments paramétrés. Suite à ces hypothèses, nos suppositions théoriques sont validées avec les données expérimentales et le calcul de l'erreur relative est mis en évidence. Les résultats du cœur #1 sont présentés.

4.3 Équations mathématiques de modélisation

Rappelons les définitions suivantes :

- $T_{exéc}$: temps d'exécution total mesuré du programme (seconde)
- B_{ext} : nombre d'itérations de la boucle externe (i)
- B_{int} : nombre d'itérations de la boucle interne (j)
- N_{mac} : nombre d'instructions « MAC » contenues dans la boucle interne
- M_0 : temps d'exécution d'un cycle de B_{ext}
- M_1 : temps d'exécution d'un cycle de B_{int}
- M_2 : temps d'exécution d'une instruction

Les résultats de la paramétrisation du programme à doubles boucles imbriquées ont été illustrés graphiquement aux figures 3.2, 3.3 et 3.4 (chapitre précédent). De ces résultats le modèle mathématique suivant est proposé :

$$T_{exéc} = B_{ext} * M_0 + B_0 \tag{4.1}$$

$$M_0 = B_{int} * M_1 + B_1 \tag{4.2}$$

$$M_1 = N_{mac} * M_2 + B_2 \tag{4.3}$$

En rassemblant le tout on obtient :

$$T_{exéc} = B_{ext} * (B_{int} * M_1 + B_1) + B_0$$

$$T_{exéc} = (B_{ext} * M_1) * B_{int} + (B_{ext} * B_1 + B_0) \tag{4.4}$$

$$T_{exéc} = B_{ext} * B_{int} * (N_{mac} * M_2 + B_2) + (B_{ext} * B_1 + B_0)$$

$$T_{exéc} = (B_{ext} * B_{int} * M_2) * N_{mac} + (B_{ext} * B_{int} * B_2 + B_{ext} * B_1 + B_0) \tag{4.5}$$

La première caractérisation, qui vise à déterminer le temps moyen d'exécution des instructions « MAC », se fait à partir du tableau 3.1 (et du graphique correspondant de la figure 3.2) et de l'équation 4.5. En fixant B_{ext} et B_{int}, nous obtenons une droite, dont la pente, $Pente_1$, est définie :

$$Pente_1 = (B_{ext} * B_{int} * M_2) \tag{4.6}$$

L'ordonnée à l'origine, $Ordonnée_1$, peut être définie :

$$Ordonnée_1 = (B_{ext} * B_{int} * B_2 + B_{ext} * B_1 + B_0). \tag{4.7}$$

En nous basant sur l'équation 4.6, le principal paramètre d'intérêt de cette caractérisation, M_2, qui représente le temps moyen d'exécution d'une instruction, devient alors:

$$M_2 = Pente_1 / (B_{ext} * B_{int}) \quad (4.8)$$

La valeur des paramètres $Pente_1$ et M_2 est estimé à partir des données du tableau 3.1 qui sont reproduites en partie dans le tableau 4.1. En utilisant un chiffrier électronique, les résultats suivants sont obtenus:

Tableau 4.1 Estimation des valeurs $Pente_1$ et M_2

Item	**N_{mac}**	**B_{ext}**	**B_{int}**	**$T_{exéc}$ (secondes)**	**$Pente_1$ $\Delta T_{exéc}/\Delta N_{mac}$ (sec/ N_{mac})**	**M_2 Équation 4.8 (sec/ B_{int*} B_{ext*} N_{mac})**
1	100	1200	65536	24.57	0.245	3.12E-09
2	50	1200	65536	12.3	0.245	3.12E-09
3	25	1200	65536	6.16	0.246	3.12E-09
4	10	1200	65536	2.47	0.246	3.12E-09
5	5	1200	65536	1.25	0.146	
6	0	1200	65536	0.52		
		fréquence MAC = 1/M2 =			3.20E+08	Hz

Mentionnons que la $Pente_1$, de l'équation 4.6 se calcule à partir du tableau 4.1 comme suit :

$$Pente_1 = \Delta T_{exéc} / \Delta N_{mac} \quad (4.9)$$

L'estimation du temps moyen d'exécution de l'instruction «MAC», M_2, permet d'en estimer la fréquence d'exécution qui sera utilisé ultérieurement.

Période d'exécution de N_{mac} $= \Delta T_{exéc}$ */ nombre d'exécution de l'instruction «MAC»*

$$= \Delta T_{exéc} / (\Delta N_{mac} * B_{ext} * B_{int})$$

$$= Pente_1 / (B_{ext} * B_{int})$$

$$= M_2$$

Donc, l'inverse de la période d'exécution de N_{mac}, définie la fréquence d'exécution de N_{mac}. Cette fréquence est donnée par la relation suivante :

$$\boxed{\textit{Fréquence d'exécution de l'instruction MAC} = 1/M_2} \qquad (4.10)$$

La seconde caractérisation, visant les paramètres liés à la boucle interne, se fait à partir du tableau 3.2 (et du graphique correspondant de la figure 3.3) et de l'équation 4.4. En fixant B_{ext} et N_{mac}, une droite est obtenue, avec comme variable indépendante B_{int} et variable dépendante le temps d'exécution du programme, pour les valeurs de $B_{int} > 64$, tel qu'illustré au tableau 4.2. La pente, $Pente_2$, de cette droite devient :

$$Pente_2 = (B_{ext} * M_1) \qquad (4.11)$$

Le paramètre M_1, qui représente le temps moyen d'exécution de chaque boucle interne, devient alors :

$$M_1 = Pente_2 / B_{ext} \qquad (4.12)$$

Les valeurs des paramètres $Pente_2$ et M_1 sont estimées en utilisant un chiffrier électronique à partir des données du tableau 3.2 reproduites dans le tableau 4.2.

Tableau 4.2 Estimation des valeurs $Pente_2$ et M_1

Item	**N_{mac}**	**B_{ext}**	**B_{int}**	**$T_{exéc}$ (secondes)**	**$Pente_2$ $\Delta T_{exéc}/\Delta B_{int}$ (sec/ B_{int})**	**M_1 Équation 4.12 (sec/ B_{int*} B_{ext})**
1	50	1200	65536	12.300637	1.88E-04	1.56E-07
2	50	1200	32768	6.151802	1.88E-04	1.56E-07
3	50	1200	16384	3.076576	1.88E-04	1.56E-07
4	50	1200	8192	1.539123	1.88E-04	1.57E-07
5	50	1200	4096	0.769527	1.88E-04	1.56E-07
6	50	1200	2048	0.385107	1.88E-04	1.56E-07
7	50	1200	1024	0.192861	1.88E-04	1.56E-07
8	50	1200	512	0.096796	1.88E-04	1.57E-07
9	50	1200	256	0.048610	1.86E-04	1.55E-07
10	50	1200	128	0.024835	1.90E-04	1.58E-07
11	50	1200	64	0.012669	1.97E-04	1.64E-07
12	50	1200	32	0.006367	1.95E-04	1.62E-07
13	50	1200	16	0.003251	1.96E-04	1.63E-07
14	50	1200	8	0.001687	1.97E-04	1.64E-07
15	50	1200	4	0.000899	1.96E-04	1.63E-07
			fréquence boucle interne = $1/M_1$ =			6.39E+06 Hz

Mentionnons que la valeur $Pente_2$ est estimée comme suit :

$$Pente_2 = \Delta T_{exéc} / \Delta B_{int} \tag{4.13}$$

L'estimation du temps d'exécution d'une boucle interne nous permet d'estimer la fréquence d'exécution de cette boucle qui sera utilisée ultérieurement. Elle est donnée par la relation suivante :

$$\textit{Fréquence d'exécution boucle interne} = 1/M_1 \tag{4.7}$$

La troisième caractérisation, celle de la boucle externe, se fait à partir du tableau 3.3, du graphique correspondant de la figure 3.4, et de l'équation 4.1. En fixant B_{int} et N_{mac}, une droite est obtenue avec comme variable indépendante B_{ext} et variable dépendante le temps d'exécution du programme.

La pente, M_0, de l'équation 4.1 de cette droite est fonction de B_{int} et N_{mac} alors que l'ordonnée à l'origine B_0 est, elle, indépendante de ces paramètres. Les valeurs de M_0 et B_0 sont estimées à partir des données du tableau 3.3, reproduites en partie au tableau 4.3. La valeur de B_1 (équation 4.2) y est également estimée.

En utilisant un chiffrier électronique, les résultats suivants sont obtenus:

Tableau 4.3 Valeurs estimées de M_0, B_0 et B_1

Item	**N_{mac}**	**B_{ext}**	**B_{int}**	**$T_{exéc}$ (secondes)**	**M_0 $\Delta T_{exé} / \Delta B_{ext}$ (sec/ B_{ext})**	**B_0 (secondes)**	**B_1 Équation 4.2 (sec/ B_{ext})**
1	50	8192	65536	83.945856	1.02E-02	2.28E-03	-1.04E-06
2	50	4096	65536	41.974070	1.02E-02	5.01E-03	-2.37E-06
3	50	2048	65536	20.990899	1.02E-02	4.26E-03	5.29E-07
4	50	1024	65536	10.496346	1.02E-02	3.48E-03	1.23E-06
5	50	512	65536	5.248713	1.03E-02	2.62E-03	5.15E-06
6	50	256	65536	2.623890	1.02E-02	2.11E-03	1.17E-08
7	50	128	65536	1.312138	1.02E-02	1.92E-03	-1.08E-05
8	50	64	65536	0.656953	1.03E-02	1.72E-03	1.16E-05
9	50	32	65536	0.328644	1.03E-02	1.55E-03	1.46E-05
10	50	16	65536	0.164441	1.03E-02	1.41E-03	2.60E-06
11	50	8	65536	0.082436	1.03E-02	1.29E-03	2.79E-05
12	50	4	65536	0.041332	1.03E-02	1.20E-03	1.29E-05
13	50	2	65536	0.020810	1.02E-02	1.12E-03	-5.06E-05
14	50	1	65536	0.010613	1.06E-02	1.03E-03	3.65E-04
15	50	0	65536	8.00E-08			
			fréquence boucle externe = $1/M_0$ =			97.56	Hz

Mentionnons que la pente M_0 et la valeur de B_0, de l'équation 4.1, se calculent à partir du tableau 4.3 comme suit :

$$M_0 = \Delta T_{exéc} / \Delta B_{ext} \quad (4.15)$$

$$B_0 = (=INTERCEPT (I\$12:I13, D\$12:D13)) \text{ fonction du chiffrier électronique} \quad (4.16)$$

Mentionnons également que M_0 représente le temps moyen d'exécution de la boucle externe. La fréquence d'exécution des boucles externes, utilisée ultérieurement est donnée par la relation suivante :

$$\text{Fréquence d'exécution boucle externe} = 1/M_0 \quad (4.17)$$

4.4 Application des équations aux données

Suite aux diverses caractérisations effectuées, il a été possible d'estimer les valeurs des paramètres de modélisation M_0, M_1, M_2, B_0, B_1 et B_2 que nous résumons dans le tableau 4.4 ci-dessous.

Tableau 4.4 Résumé des valeurs estimées des paramètres de modélisation

M_0	M_1	M_2	B_0	B_1	B_2
1.03E-02	1.56E-07	3.12E-09	2.21E-03	2.69E-05	3.29E-10

Rappelons que toutes ces valeurs ont été estimées dans la partie linéaire des différentes courbes de caractérisation. En utilisant l'équation 4.5, avec les paramètres du tableau 4.4, le temps théorique d'exécution des programmes des tableaux 3.1, 3.2 et 3.3 est estimé. De même que l'erreur relative entre les temps d'exécution total mesuré et théorique.

Tableau 4.5 Erreur relative – Modélisation instruction « MAC »

Item	**N_{mac}**	**B_{ext}**	**B_{int}**	**$T_{exéc}$ (secondes)**	**Temps Exécution Théorique (secondes)**	**Erreur Relative (%)**
1	100	1200	65536	24.57	24.61	-0.15
2	50	1200	65536	12.3	12.33	-0.31
3	25	1200	65536	6.16	6.20	-0.60
4	10	1200	65536	2.47	2.52	-1.65
5	5	1200	65536	1.25	1.29	-3.31
6	0	1200	65536	0.52	0.06	88.35

Tableau 4.6 Erreur relative – Modélisation boucle interne

Item	N_{mac}	B_{ext}	B_{int}	$T_{exéc}$ **(secondes)**	Temps **Exécution Théorique (secondes)**	Erreur **Relative (%)**
1	50	1200	65536	12.30	12.33	-0.28
2	50	1200	32768	6.15	6.18	-0.54
3	50	1200	16384	3.08	3.11	-1.07
4	50	1200	8192	1.54	1.57	-2.14
5	50	1200	4096	0.77	0.80	-4.38
6	50	1200	2048	0.39	0.42	-8.77
7	50	1200	1024	0.19	0.23	-17.54
8	50	1200	512	0.097	0.13	-34.91
9	50	1200	256	0.049	0.08	-69.79
10	50	1200	128	0.025	0.06	-135.61
11	50	1200	64	0.013	0.05	-267.05
12	50	1200	32	0.006	0.04	-536.05
13	50	1200	16	0.0033	0.04	-1053.31
14	50	1200	8	0.0017	0.04	-2033.66
15	50	1200	4	0.0009	0.04	-3821.20
16	50	1200	2	0.0005	0.03	-6784.06
17	50	1200	1	0.0003	0.03	-10958.8
18	50	1200	0	0.000082		

Le tableau 4.5 illustre que le modèle mathématique est fiable pour $N_{mac} \geq 5$, et très précis pour $N_{mac} \geq 25$. Le tableau 4.6 illustre que le modèle mathématique est fiable pour $B_{int} \geq 2048$ et très précis pour $B_{int} \geq 16384$.

Tableau 4.7 Erreur relative – Modélisation boucle externe

Item	**N_{mac}**	**B_{ext}**	**B_{int}**	**$T_{exéc}$ (secondes)**	**Temps Exécution Théorique (secondes)**	**Erreur Relative (%)**
1	50	8192	65536	83.95	84.19	-0.30
2	50	4096	65536	41.97	42.10	-0.30
3	50	2048	65536	20.99	21.05	-0.28
4	50	1024	65536	10.50	10.53	-0.28
5	50	512	65536	5.25	5.26	-0.29
6	50	256	65536	2.62	2.63	-0.35
7	50	128	65536	1.31	1.32	-0.42
8	50	64	65536	0.66	0.66	-0.46
9	50	32	65536	0.33	0.33	-0.74
10	50	16	65536	0.16	0.17	-1.34
11	50	8	65536	0.082	0.084	-2.42
12	50	4	65536	0.041	0.043	-4.82
13	50	2	65536	0.021	0.023	-9.41
14	50	1	65536	0.011	0.012	-17.70
15	50	0	65536	8.00E-08		

Le tableau 4.7 montre que le modèle mathématique est fiable pour $B_{ext} \geq 4$ et très précis pour $B_{ext} \geq 32$.

4.5 Conclusion

Dans ce chapitre, la modélisation avec des équations mathématiques de la paramétrisation d'un programme à 2 boucles imbriquées a été effectuée. Cette modélisation nous amène à calculer une vitesse d'exécution distincte ainsi qu'un temps d'exécution propre de chaque instruction d'un processeur asynchrone. La fiabilité de notre modèle mathématique a été constatée en comparant le temps d'exécution mesuré au temps d'exécution théorique calculé.

Dans chacun des cas, une excellente précision a été observée avec un grand nombre d'itérations ou d'instructions exécutées.

CHAPITRE 5

IDENTIFICATION ET ESTIMATION DES FRÉQUENCES D'EXÉCUTION DES INSTRUCTIONS

5.1 Introduction

Un programme à 2 boucles imbriquées a été modélisé dans le chapitre précédent. Le temps d'exécution d'une instruction « MAC » (M_2) a été calculé, ainsi que les temps d'exécution des boucles interne (M_1) et externe (M_0) du programme. Ce programme fonctionnant en boucle provoque une demande de courant à la source d'alimentation à chaque fois qu'une instruction est exécutée. Ceci se produisant de façon répétitive, posons l'hypothèse que cette répétition va se traduire par l'apparition, dans le spectre fréquentiel de la consommation de courant, de la fréquence d'exécution de l'instruction ($1/M_2$) et des boucles internes ($1/M_1$) et externes ($1/M_0$). Ce chapitre porte sur la validation de cette hypothèse, à l'aide d'un analyseur de spectres.

5.2 Méthodologie

La méthodologie consiste à exécuter un programme utilisant la même instruction « ADD » à l'intérieur de deux boucles imbriquées en variant les paramètres d'itérations de boucle et du nombre d'instructions dans la boucle. Le code utilisé est du type de celui de la figure 5.1. Le programme exécuté a été modélisé de la même façon qu'au chapitre précédent. La fréquence d'exécution de l'instruction « ADD » ($1/M_2$) ainsi que celles des boucles internes ($1/M_1$) et externes ($1/M_0$) sont calculées. Chacune des fréquences d'exécution est validée par une capture d'écran de l'analyseur de spectres.

```
_asm rdtsc timerStart;

register unsigned int Val1 = 1;
register unsigned int Val2 = 1;
register unsigned int Val3 = 0;

for(i=0; i<600; i++)
   {
       for(j=0; j<65536; j++)
         {
            _asm add Val3, Val1, Val2;
         }
   }

_asm
   {
       rdtsc timerStop;
       sub timerDiff, timerStart, timerStop;
   }
```

Figure 5.1 Exemple de programme à doubles boucles imbriquées

5.3 Identification et estimation des fréquences d'exécution de l'instruction « ADD » pour un seul cœur

Cette section contient les captures d'écrans de l'analyseur de spectres montrant les fréquences d'exécution de l'instruction « ADD » et des boucles interne et externe du programme étudié. Les résultats spécifiques du cœur #9 sont présentés.

5.3.1 Identification et estimation de la fréquence d'exécution de l'instruction « ADD »

Les figures 5.2 et 5.3 illustrent les captures d'écran de l'analyseur de spectre avant et après l'exécution du programme de doubles boucles imbriquées de la figure 5.1 utilisant l'instruction « ADD ». La figure 5.3 montre très clairement l'apparition d'une fréquence causée par la demande en courant de façon répétitive de l'exécution de l'instruction « ADD ».

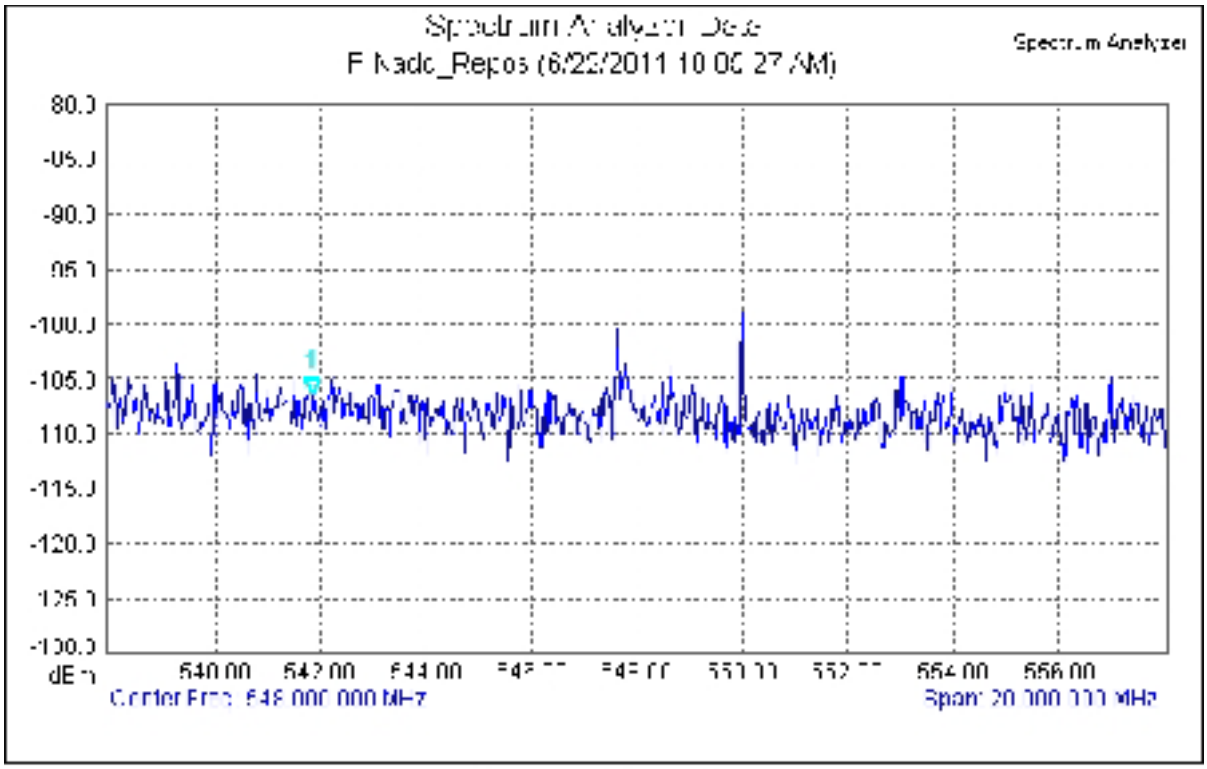


Figure 5.2 « ADD » avant l'exécution du programme, notez la position du marqueur

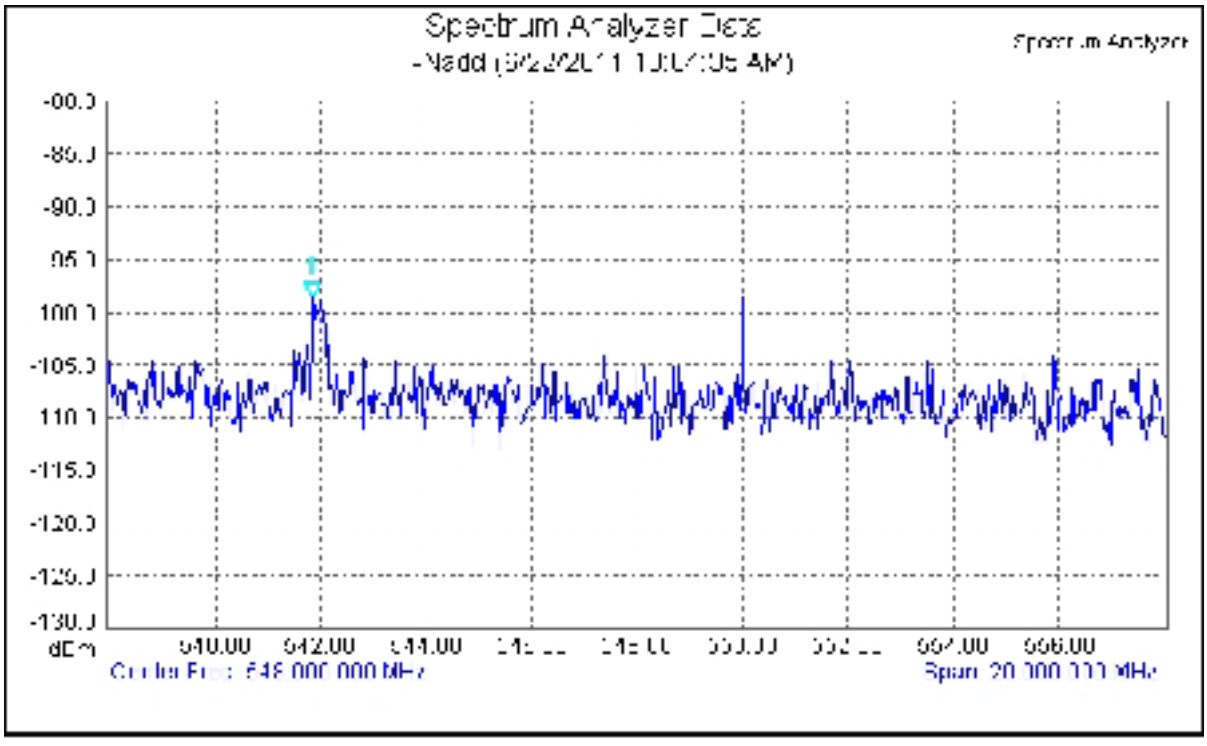


Figure 5.3 « ADD » après l'exécution du programme, notez la position du marqueur

Notre modèle prédisait une fréquence d'exécution de l'instruction « ADD » de 544.64 MHz. Nous avons observé à l'analyseur de spectres, une fréquence de 541.8 MHz. Ceci donne une erreur relative de 0.5%.

5.3.2 Identification et estimation de la fréquence d'exécution de la boucle interne

Les figures 5.4 et 5.5 illustrent les captures d'écran de l'analyseur de spectre avant et après l'exécution du programme de doubles boucles imbriquées de la figure 5.1 contenant l'instruction « ADD ». La figure 5.5 montre très clairement l'apparition d'une fréquence causée par la demande en courant de façon répétitive de l'exécution de la boucle interne du programme.

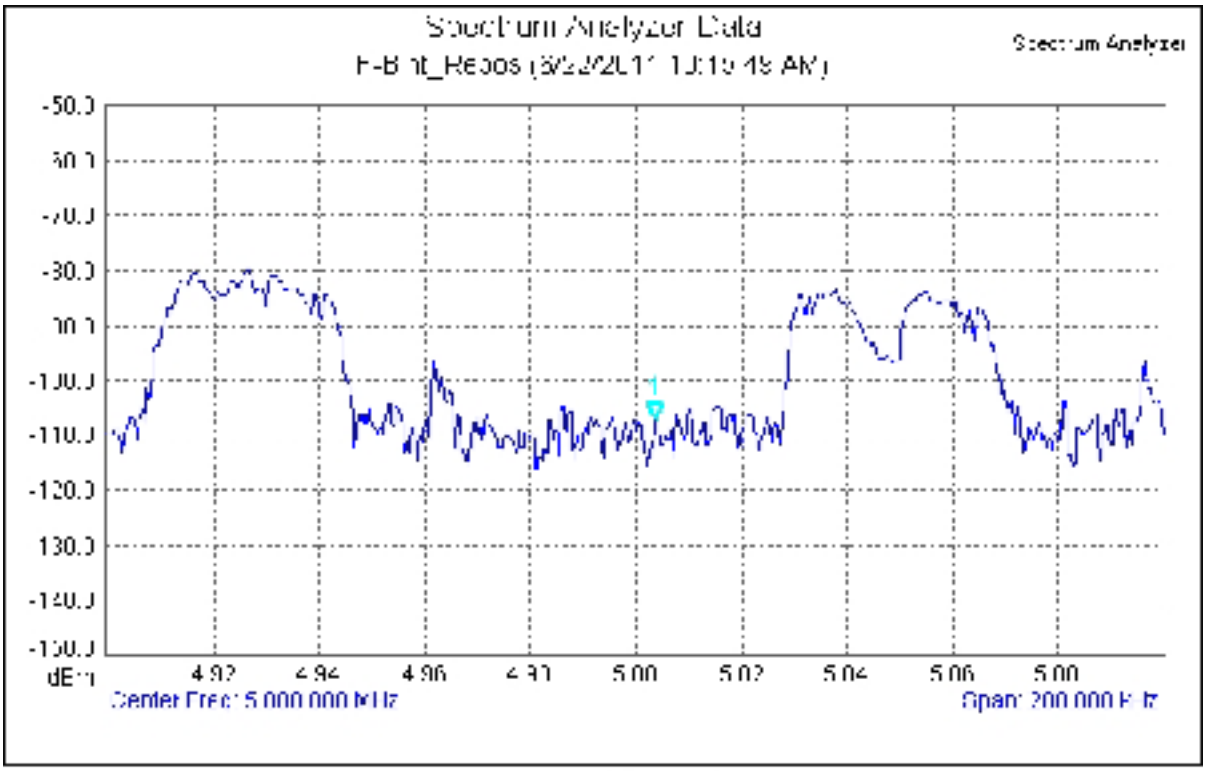


Figure 5.4 Boucle interne avant l'exécution du programme, notez la position du marqueur

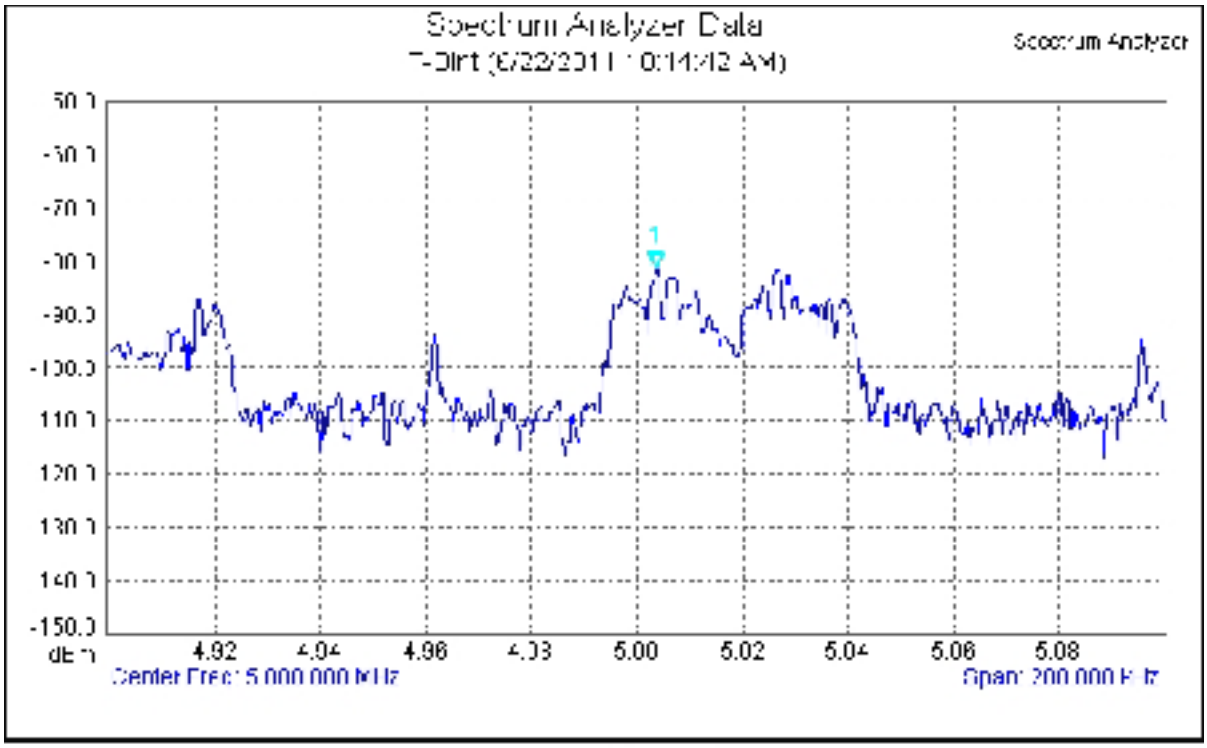


Figure 5.5 Boucle interne après l'exécution du programme, notez la position du marqueur

Notre modèle prédisait une fréquence d'exécution de la boucle interne de 5.06 MHz. Nous avons observé à l'analyseur de spectres, une fréquence de 5.0 MHz. Ceci donne une erreur relative de 1.2%.

5.3.3 Identification et estimation de la fréquence d'exécution de la boucle externe

Les figures 5.6 et 5.7 illustrent les captures d'écran de l'analyseur de spectre avant et après l'exécution du programme de doubles boucles imbriquées de la figure 5.1 contenant l'instruction « ADD ». La figure 5.7 montre très clairement l'apparition d'une fréquence causée par la demande en courant de façon répétitive de l'exécution de la boucle externe du programme.

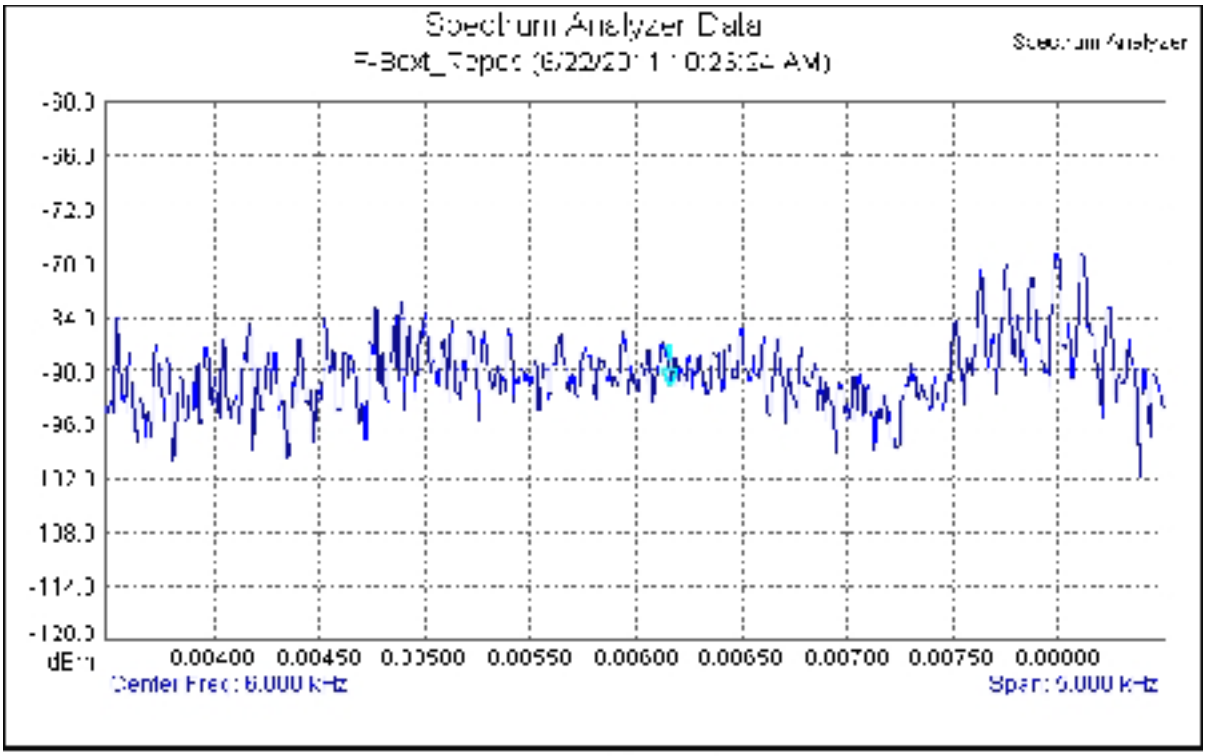


Figure 5.6 Boucle externe avant l'exécution du programme, notez la position du marqueur

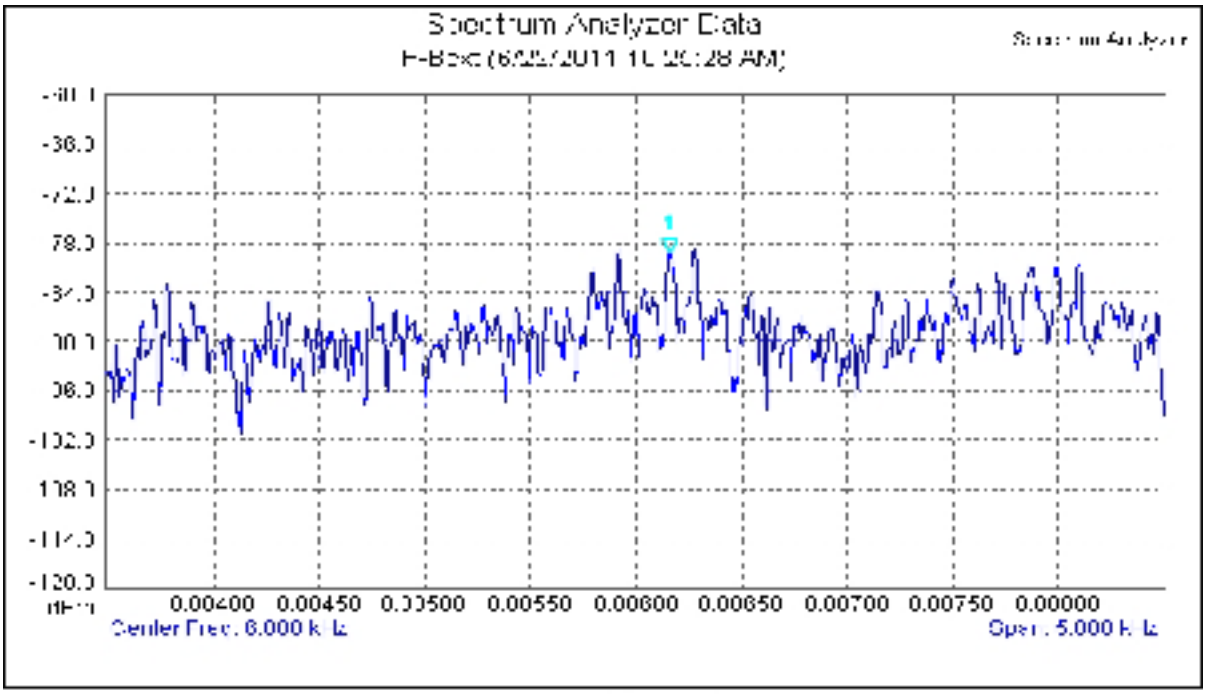


Figure 5.7 Boucle externe après l'exécution du programme, notez la position du marqueur

Notre modèle prédisait une fréquence d'exécution de la boucle externe de 3.10 KHz. La seconde harmonique de notre modèle a une fréquence de 6.21 KHz. Nous avons observé à l'analyseur de spectres, une fréquence de 6.2 KHz. Ceci donne une erreur relative de 0.2% sur la seconde harmonique.

Il est très important de noter que c'est la seconde harmonique de notre modèle qui a été observée et non la fréquence fondamentale. À ce jour, nous n'avons pas d'explication à ce phénomène. Plusieurs hypothèses sont envisagées, dont l'influence des mécanismes internes de transfert de données dans le contexte de l'insuffisance de ressources internes des cœurs mais la validation de celles-ci dépasse le cadre de ce travail.

5.4 Identification et estimation des fréquences d'exécution de l'instruction « ADD » pour l'ensemble des cœurs

Les sections précédentes ont présenté les captures d'écrans de fréquences d'exécution produites par l'instruction « ADD » et de ses boucles imbriquées pour le cœur #9. Cette section présente un tableau de toutes les fréquences d'exécution pour l'instruction « ADD » de l'ensemble des cœurs. Le programme de la figure 5.1 est exécuté.

Les tableaux 5.1, 5.2 et 5.3 illustrent les fréquences d'exécution de l'instruction « ADD », de la boucle interne et de la boucle externe pour l'ensemble des quinze cœurs. Le tableau 5.1 donne une erreur relative variant de 0.1% à 1.3%. Le tableau 5.2 présente une erreur relative variant de 0.03% à 2.4%.

Tableau 5.1 Fréquences d'exécution de l'instruction « ADD »

	Fréquences Instruction ADD		
	Fréquence Analyseur	Fréquence Modèle	Erreur Relative
Cœur #	(MHz)	(MHz)	
1	541.85455	541.39	0.1%
2	541.70909	548.69	-1.3%
3	542.07273	544.29	-0.4%
4	541.85455	543.30	-0.3%
5	541.70909	548.75	-1.3%
6	542.14545	543.19	-0.2%
7	541.96364	543.08	-0.2%
8	542.25455	549.31	-1.3%
9	541.85455	544.64	-0.5%
10	541.89091	540.45	0.3%
11	542.21818	546.47	-0.8%
12	542.72727	539.60	0.6%
13	542.83636	537.30	1.0%
14	541.96364	543.83	-0.3%
15	541.92727	537.16	0.9%

Tableau 5.2 Fréquences d'exécution de la boucle interne

Cœur #	Fréquences Boucle Interne		
	Fréquence Analyseur (MHz)	Fréquence Modèle (MHz)	Erreur Relative
1	4.989818	5.0359	-0.9%
2	4.993454	5.0980	-2.1%
3	5.012363	5.0668	-1.1%
4	5.023273	5.0517	-0.6%
5	4.981818	5.0996	-2.4%
6	5.011636	5.0531	-0.8%
7	4.987636	5.0506	-1.3%
8	5.032000	5.1118	-1.6%
9	5.003636	5.0650	-1.2%
10	5.014545	5.0256	-0.2%
11	5.025090	5.0820	-1.1%
12	5.025090	5.0238	0.03%
13	5.024727	5.0039	0.4%
14	4.984727	5.0574	-1.5%
15	5.002545	4.9897	0.3%

Tableau 5.3 Fréquences d'exécution de la boucle externe

Cœur #	Fréquence Boucle Externe			
	Fréquence Analyseur (KHz)	Fréquence Modèle (KHz)	2ième Harm. Modèle (KHz)	Erreur Relative
1	6.163	3.08864	6.17728	-0.2%
2	6.245	3.13145	6.26290	-0.3%
3	6.227	3.11067	6.22134	0.1%
4	6.227	3.09983	6.19966	0.4%
5	6.209	3.12705	6.25410	-0.7%
6	6.136	3.10089	6.20178	-1.1%
7	6.163	3.09806	6.19612	-0.5%
8	6.209	3.13711	6.27422	-1.1%
9	6.163	3.10732	6.21464	-0.8%
10	6.136	3.08257	6.16514	-0.5%
11	6.254	3.11714	6.23428	0.3%
12	6.109	3.08315	6.16630	-0.9%
13	6.181	3.07131	6.14262	0.6%
14	6.127	3.10566	6.21132	-1.4%
15	6.109	3.06244	6.12488	-0.3%

Le tableau 5.3 illustre l'erreur relative variant de 0.1% à 1.4% sur la deuxième harmonique de la fréquence calculée de notre modèle.

5.5 Conclusion

Dans ce chapitre, il a été clairement montré qu'il est possible de produire des fréquences d'exécution d'instructions et de boucles en exécutant un programme en boucle. La période de ces fréquences d'exécution représente le temps d'exécution propre de ces instructions pour un processeur asynchrone. L'hypothèse selon laquelle un programme fonctionnant en boucle provoque une demande de courant à la source d'alimentation à chaque fois qu'une instruction est exécutée et que cette répétition va se traduire par l'apparition, dans le spectre fréquentiel de la consommation de courant, de la fréquence d'exécution de l'instruction ($1/M_2$) et des boucles internes ($1/M_1$) et externes ($1/M_0$) est donc validée. Concernant la boucle externe de notre programme, le modèle indique une fréquence spécifique d'exécution

alors que l'expérience reproduit sa deuxième harmonique. À ce jour, nous n'avons pas d'explication à ce phénomène. Rappelons que plusieurs hypothèses sont envisagées, dont l'influence des mécanismes internes de transfert de données dans le contexte de l'insuffisance de ressources internes des cœurs mais la validation de celles-ci dépasse le cadre de ce travail.

CHAPITRE 6

CARACTÉRISATION DES INSTRUCTIONS D'UN FILTRE « FIR »

6.1 Introduction

L'objectif final de cette recherche consiste à faciliter la diminution de la consommation énergétique liée à l'exécution de programmes informatiques en télécommunication, via l'estimation hâtive de cette consommation. Les programmes de filtres « FIR » en sont un exemple typique. L'annexe II reproduit le code d'un programme simple de filtre « FIR ». En caractérisant la consommation énergétique des instructions de l'annexe II, il devient possible de déterminer lesquelles sont moins énergivores et ainsi guider la conception future de ces filtres.

6.2 Méthodologie

La même méthodologie qu'au chapitre 4 est utilisée afin de trouver le temps d'exécution des différentes instructions (M_2) ainsi que leur fréquence d'exécution propre ($1/M_2$). Le même type de programme avec 2 boucles imbriquées que celui de la figure 3.1 est exécuté. Les instructions suivantes se retrouvent toutes dans le programme typique d'un filtre « FIR » présenté à l'annexe II:

- MOV
- MAC.LL
- SHL
- SHR
- NOP
- IF
- FOR
- DIV
- **Assig. #1:** assignation de type **Register signed_int32 VAR = 1**;
- **Assig. #2:** assignation de type **Vecteur1 [j] = Vecteur2 [j]**;
- **Assig. #3:** assignation de type **Val1 = Vecteur2 [j]**;
- **Assig. #4:** assignation de type **Vecteur1 [j] = VAL1**;

6.3 Caractéristiques fréquentielles

Le tableau 6.1 dresse la liste des caractéristiques fréquentielles des instructions étudiées.

Tableau 6.1 Caractéristiques fréquentielles - Instructions

Instr.	Fréquence Exécution Théorique (Hz)	Temps Exécution Théorique (sec)
MOV	9.61E+08	1.04E-09
MAC	3.21E+08	3.12E-09
SHL	5.62E+08	1.78E-09
SHR	5.62E+08	1.78E-09
NOP	9.40E+08	1.06E-09
DIV	4.09E+07	2.44E-08
IF	2.26E+07	4.43E-08
FOR	1.46E+07	6.86E-08
Assig. #1	9.60E+08	1.04E-09
Assig. #2	5.26E+07	1.90E-08
Assig. #3	1.71E+08	5.86E-09
Assig. #4	1.56E+08	6.39E-09

Le tableau 6.1 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées telles que décrites à la section 6.2;
2. **Fréquence Exécution Théorique** : fréquence d'exécution, $1/M_2$, d'une seule instruction calculée à l'aide du modèle mathématique du chapitre 4;
3. **Temps Exécution Théorique**: temps d'exécution, M_2, correspondant à la période de la **Fréquence Exécution Théorique**.

Le tableau 6.1 présente la fréquence d'exécution théorique des instructions lorsque celles-ci sont exécutées en boucle ainsi que le temps d'exécution théorique de chacune d'entre elles.

6.4 Caractéristiques de puissance

Le tableau 6.2 dresse la liste de toutes les mesures prises afin de compléter le tableau 6.3 énumérant les caractéristiques de puissance des diverses instructions.

Tableau 6.2 Prise de mesures - Caractéristiques de puissance - Instructions

	Sortie Ampli		Point de Test				
Instr.	Tension 15 cœurs Mesurée (Vrms)	Gain Ampli Mesuré	Tension 15 cœurs (Vrms)	R (Ohm)	Courant 15 cœurs (Irms)	Courant Moyen par cœur (Irms)	Tension Cœur Mesurée (Vrms)
MOV	1.056	8670	1.22E-04	0.005	2.44E-02	1.62E-03	1.167
MAC	1.091	8670	1.26E-04	0.005	2.52E-02	1.68E-03	1.167
SHL	1.061	8670	1.22E-04	0.005	2.45E-02	1.63E-03	1.167
SHR	1.065	8670	1.23E-04	0.005	2.46E-02	1.64E-03	1.167
NOP	1.066	8670	1.23E-04	0.005	2.46E-02	1.64E-03	1.167
DIV	1.032	8670	1.19E-04	0.005	2.38E-02	1.59E-03	1.167
IF	1.062	8670	1.22E-04	0.005	2.45E-02	1.63E-03	1.167
FOR	1.049	8670	1.21E-04	0.005	2.42E-02	1.61E-03	1.167
Assig. #1	1.052	8670	1.21E-04	0.005	2.43E-02	1.62E-03	1.167
Assig. #2	1.073	8670	1.24E-04	0.005	2.47E-02	1.65E-03	1.167
Assig. #3	1.049	8670	1.21E-04	0.005	2.42E-02	1.61E-03	1.167
Assig. #4	1.047	8670	1.21E-04	0.005	2.42E-02	1.61E-03	1.167

Le tableau 6.2 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées telles que décrites à la section 6.2;
2. **Sortie Ampli**: mesures prises à la sortie de l'amplificateur tel que décrit à la section 2.3;
3. **Tension 15 cœurs Mesurée**: tension mesurée à la sortie de l'amplificateur de l'ensemble des 15 cœurs exécutant simultanément le même programme de deux boucles imbriquées contenant un seul type d'instructions;
4. **Gain Ampli Mesuré**: le gain de l'amplificateur mesuré en laboratoire. Ce gain est constant dans la gamme des tensions mesurées;

5. **Point de Test**: mesures prises de la résistance de 0.005 Ω du VEB tel que décrit à la section 2.2. Tous les courants des 15 cœurs passent par cette résistance;
6. **Tension 15 cœurs: Tension 15 cœurs Mesurée** divisé par le **Gain Ampli Mesuré.** Donne la tension des 15 cœurs au **Point de Test**;
7. **R**: résistance de 0.005 Ohm sur la plaquette au travers de laquelle tous les courants des 15 cœurs passent;
8. **Courant 15 cœurs**: **Tension 15 cœurs** divisé par la résistance **R.** Donne le courant total des 15 cœurs exécutant le programme;
9. **Courant Moyen par cœur: Courant 15 cœurs** divisé par 15. Donne le courant utilisé pour un seul cœur exécutant le programme;
10. **Tension Cœur Mesurée**: tension alimentant les cœurs mesurés en laboratoire. Les spécifications du manufacturier indiquent: Min 1.11 V, TYP 1.16 V, MAX 1.21 V;

Tableau 6.3 Caractéristiques de puissance - Instructions

Instr.	Puissance Moyenne par cœur (W)	Puissance Moyenne par «ALU» (W)
MOV	1.90E-03	1.18E-04
MAC	1.96E-03	1.22E-04
SHL	1.90E-03	1.19E-04
SHR	1.91E-03	1.19E-04
NOP	1.91E-03	1.20E-04
DIV	1.85E-03	1.16E-04
IF	1.91E-03	1.19E-04
FOR	1.88E-03	1.18E-04
Assig. #1	1.89E-03	1.18E-04
Assig. #2	1.93E-03	1.20E-04
Assig. #3	1.88E-03	1.18E-04
Assig. #4	1.88E-03	1.17E-04

Le tableau 6.3 se compose des divers éléments suivants calculés à partir du tableau 6.2:

1. **Instr.**: les instructions étudiées telles que décrites à la section 6.2;
2. **Puissance Moyenne par cœur**: **Courant Moyen par cœur** multiplié par la **Tension Cœur Mesurée.** Donne la puissance moyenne consommée pour chaque cœur;
3. **Puissance Moyenne par « ALU »**: **Puissance Moyenne par cœur** divisée par 16. Donne la puissance moyenne consommée pour chaque « ALU ».

Le tableau 6.3 dresse la liste de la puissance moyenne consommée par cœur pour chaque type d'instructions. Cette puissance varie selon le type d'instructions de 188 mW à 193 mW. Le tableau 6.3 dénote également la puissance moyenne consommée par chaque « ALU » pour chaque type d'instructions.

6.5 Caractéristiques énergétiques

Le tableau 6.4 dresse la liste des diverses mesures prises afin de compléter le tableau 6.5 qui présente les caractéristiques énergétiques par type d'instructions.

Tableau 6.4 Prise de mesures - Caractéristiques énergétiques – Instructions

Instr.	Nombre instr.	B_{ext}	B_{int}	Total instr.	Courant Moyen par cœur (Irms)	Tension Cœur Mesurée (Vrms)	$T_{exéc}$ (cœur #1) (sec)
MOV	1000	256	65536	1.68E+10	1.62E-03	1.167	17.66
MAC	1000	256	65536	1.68E+10	1.68E-03	1.167	52.35
SHL	1000	256	65536	1.68E+10	1.63E-03	1.167	29.88
SHR	1000	256	65536	1.68E+10	1.64E-03	1.167	29.88
NOP	1000	256	65536	1.68E+10	1.64E-03	1.167	18.03
DIV	700	64	8192	3.67E+08	1.59E-03	1.167	8.97
IF	700	64	8192	3.67E+08	1.63E-03	1.167	16.27
FOR	700	64	8192	3.67E+08	1.61E-03	1.167	25.19
Assig. #1	45	4096	65536	1.21E+10	1.62E-03	1.167	15.57
Assig. #2	1000	1024	1024	1.05E+09	1.65E-03	1.167	18.8
Assig. #3	1000	1024	1024	1.05E+09	1.61E-03	1.167	6.15
Assig. #4	1000	2048	1024	2.10E+09	1.61E-03	1.167	13.4

Le tableau 6.4 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées;
2. **Nombre instr.**: le nombre d'instructions à l'intérieur de la deuxième boucle imbriquée;
3. $\mathbf{B_{ext}}$: le nombre d'itérations que la boucle externe effectue;
4. $\mathbf{B_{int}}$: le nombre d'itérations que la boucle interne effectue;
5. **Total instr.**: **Nombre instr.** multiplié par $\mathbf{B_{ext}}$ multiplié par $\mathbf{B_{int}}$. Le nombre total d'instructions exécuté par le programme au travers des deux boucles imbriquées;

6. **Courant Moyen par cœur: Courant 15 cœurs** divisé par 15. Donne le courant utilisé pour un seul cœur exécutant le programme. Du tableau 6.2;
7. **Tension Cœur Mesurée**: tension alimentant les cœurs. Les spécifications du manufacturier indiquent: Min 1.11 V, TYP 1.16 V, MAX 1.21 V;
8. $\mathbf{T_{exéc}}$: le temps mesuré que met le programme pour s'exécuter avec un seul cœur; le cœur #1.

Tableau 6.5 Caractéristiques énergétiques - Instructions

Instr.	Énergie Totale Dissipée (J)	Énergie Moyenne par instr. (J)
MOV	3.35E-02	2.00E-12
MAC	1.03E-01	6.11E-12
SHL	5.69E-02	3.39E-12
SHR	5.71E-02	3.40E-12
NOP	3.45E-02	2.06E-12
DIV	1.66E-02	4.53E-11
IF	3.10E-02	8.45E-11
FOR	4.74E-02	1.29E-10
Assig. #1	2.94E-02	2.43E-12
Assig. #2	3.62E-02	3.45E-11
Assig. #3	1.16E-02	1.10E-11
Assig. #4	2.52E-02	1.20E-11

Le tableau 6.5 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées;
2. **Énergie Totale Dissipée**: **Courant Moyen par cœur** multiplié par **Tension Cœur Mesurée** multiplié par $\mathbf{T_{exéc}}$ du tableau 6.4. Donne l'énergie totale consommée pour chaque cœur;
3. **Énergie Moyenne par instr.**: **Énergie Totale Dissipée** divisée par **Total instr.** du tableau 6.4. Donne l'énergie moyenne consommée par instruction.

Le tableau 6.5 présente l'énergie moyenne consommée par chaque cœur pour chaque type d'instructions ainsi que la liste de l'énergie moyenne consommée par chaque instruction. Cette énergie varie de 2.00 pJ pour l'instruction MOV à 6.11 pJ pour l'instruction MAC. On observe également que les assignations et les instructions en C consomment en général plus d'énergie que les instructions en assembleur.

6.6 Caractéristiques temporelles

Tableau 6.6 Caractéristiques temporelles (1) – Instructions

	Total instr.	**Temps Exécution Théorique**	**Temps Total Exécution Théorique**	**$T_{exéc}$ (cœur #1)**	**Erreur Relative**
Instr.		**(sec)**	**(sec)**	**(sec)**	**(%)**
MOV	1.68E+10	1.04E-09	17.46	17.66	1.14%
MAC	1.68E+10	3.12E-09	52.3	52.35	0.10%
SHL	1.68E+10	1.78E-09	29.83	29.88	0.16%
SHR	1.68E+10	1.78E-09	29.83	29.88	0.14%
NOP	1.68E+10	1.06E-09	17.84	18.03	1.04%
DIV	3.67E+08	2.44E-08	8.96	8.97	0.11%
IF	3.67E+08	4.43E-08	16.27	16.27	0.04%
FOR	3.67E+08	6.86E-08	25.18	25.19	0.05%
Assig. #1	1.21E+10	1.04E-09	12.58	15.57	19.20%
Assig. #2	1.05E+09	1.90E-08	19.92	18.8	-5.98%
Assig. #3	1.05E+09	5.86E-09	6.14	6.15	0.03%
Assig. #4	2.10E+09	6.39E-09	13.4	13.4	-0.04%

Le tableau 6.6 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées;
2. **Total instr.**: Le nombre total d'instructions exécuté par le programme au travers des deux boucles imbriquées calculé au tableau 6.4;
3. **Temps Exécution Théorique**: temps d'exécution d'une seule instruction. La période de la **Fréquence Exécution Théorique** calculé au tableau 6.1;

4. **Temps Total Exécution Théorique: Total instr.** multiplié par **Temps Exécution Théorique**. Le temps total théorique requis pour exécuter le programme;
5. **$T_{exéc}$:** même que celui du tableau 6.4. le temps mesuré que met le programme pour s'exécuter avec un seul cœur; le cœur #1;
6. **Erreur Relative** : erreur relative entre le **Temps Total Exécution Théorique** et le **$T_{exéc}$.**

Tableau 6.7 Caractéristiques temporelles (2) – Instructions

Instr.	Énergie Moyenne par instr. (J)	Puissance Moyenne par cœur (W)	Temps Exécution par instr. (sec)	Temps Exécution Théorique (sec)	Erreur Relative (%)
MOV	2.00E-12	1.90E-03	1.05E-09	1.04E-09	1.14%
MAC	6.11E-12	1.96E-03	3.12E-09	3.12E-09	0.10%
SHL	3.39E-12	1.90E-03	1.78E-09	1.78E-09	0.16%
SHR	3.40E-12	1.91E-03	1.78E-09	1.78E-09	0.14%
NOP	2.06E-12	1.91E-03	1.07E-09	1.06E-09	1.04%
DIV	4.53E-11	1.85E-03	2.45E-08	2.44E-08	0.11%
IF	8.45E-11	1.91E-03	4.43E-08	4.43E-08	0.04%
FOR	1.29E-10	1.88E-03	6.86E-08	6.86E-08	0.05%
Assig. #1	2.43E-12	1.89E-03	1.29E-09	1.04E-09	19.20%
Assig. #2	3.45E-11	1.93E-03	1.79E-08	1.90E-08	-5.98%
Assig. #3	1.10E-11	1.88E-03	5.86E-09	5.86E-09	0.03%
Assig. #4	1.20E-11	1.88E-03	6.39E-09	6.39E-09	-0.04%

Le tableau 6.7 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées;
2. **Énergie Moyenne par instr.**: du tableau 6.5. Donne l'énergie moyenne consommée par instruction;
3. **Puissance Moyenne par cœur:** du tableau 6.3. Donne la puissance moyenne consommée pour chaque cœur;
4. **Temps Exécution par instr.: Énergie Moyenne par instr.** divisé par **Puissance Moyenne par cœur.** Donne le temps d'exécution pratique par instruction;

5. **Temps Exécution Théorique**: temps d'exécution théorique d'une seule instruction. La période de la **Fréquence Exécution Théorique** calculé au tableau 6.1;
6. **Erreur relative** : erreur relative entre le **Temps Exécution par instr.** et le **Temps Exécution Théorique**.

Le tableau 6.6 donne l'erreur relative entre le temps total d'exécution théorique que met le programme à s'exécuter et le temps réel mesuré. Sauf pour les assignations #1 et #2, les erreurs relatives sont très faibles. Ceci démontre la validité du modèle présenté au chapitre 4, particulièrement pour les instructions en assembleur. Concernant les assignations #1 et #2, nous posons l'hypothèse que ces instruction étant en langage C et non en langage machine, sont beaucoup plus complexes et pouvant comprendre des dépendances de variables, s'appliquent difficilement à notre modèle. Sur ce dernier point, une analyse plus approfondie est de mise. Cette analyse dépasse le cadre de ces travaux.

Le tableau 6.7 présente une autre façon de calculer l'erreur relative. L'énergie moyenne consommée par instruction et la puissance moyenne consommée pour chaque cœur sont utilisées. L'erreur relative calculée est la même que celle du tableau 6.6.

6.7 Conclusion

Dans ce chapitre, le temps propre d'exécution des instructions typiques servant à former un programme pour filtre FIR est calculé. La consommation moyenne en puissance de chaque cœur pour chaque type d'instructions est calculée. La consommation énergétique moyenne de chaque instruction est également calculée. Du temps d'exécution des instructions, il est possible de calculer le temps d'exécution théorique total de chaque programme étudié. Enfin, l'erreur relative entre le temps calculé d'exécution des programmes et leur temps réel mesuré a été montrée. Ces erreurs relatives sont faibles pour toutes les instructions en assembleur, ce qui confirme la validité du modèle mathématique du chapitre 4.

CHAPITRE 7

CARACTÉRISATION D'UN FILTRE « FIR »

7.1 Introduction

Les chapitres précédents ont montré qu'il est possible de calculer le temps d'exécution d'une instruction et d'en calculer sa consommation énergétique. Dans ce chapitre, l'application de notre modèle énergétique sur un filtre réel de type « FIR » est étudié.

7.2 Méthodologie

Le code du programme du filtre « FIR » est présenté à l'annexe II. Les caractéristiques énergétiques des instructions ont déjà été calculées au tableau 6.5. Ces données sont utilisées pour calculer la consommation énergétique théorique du filtre. Suite à ces calculs théoriques, les résultats sont vérifiés en exécutant le programme de l'annexe II et en mesurant la consommation énergétique réelle de notre filtre.

7.3 Caractéristiques énergétiques théoriques

Le tableau 7.1 illustre les caractéristiques énergétiques théoriques du filtre étudié. L'énergie totale consommée par type d'instructions est calculée. La somme de ces consommations énergétiques représente la consommation totale théorique en énergie du programme. Le temps total que met chaque type d'instructions à s'exécuter est également calculé. La somme de ces temps d'exécution représente le temps d'exécution total théorique du programme.

Tableau 7.1 Caractéristiques énergétiques théoriques du filtre

	Instr.	Total Instr.	Énergie Moyenne par instr. (J)	Énergie Totale par instr. (J)	Temps Exécution Théorique (sec)	Total Temps Exécution Théorique (sec)
	MOV	6000000	2.00E-12	1.20E-05	1.04E-09	0.00624
	MAC	54000000	6.11E-12	3.30E-04	3.12E-09	0.16834
	SHL	3000000	3.39E-12	1.02E-05	1.78E-09	0.00533
	SHR	3000000	3.40E-12	1.02E-05	1.78E-09	0.00533
	NOP	5400000	2.06E-12	1.11E-05	1.06E-09	0.00574
	DIV	60000	4.53E-11	2.72E-06	2.44E-08	0.00147
	IF	7200000	8.45E-11	6.08E-04	4.43E-08	0.31913
	Assig. #1	36000000	2.43E-12	8.76E-05	1.04E-09	0.0375
	Assig. #2	92400000	3.45E-11	3.19E-03	1.90E-08	1.75535
	Assig. #3	108000000	1.10E-11	1.19E-03	5.86E-09	0.63286
	Assig. #4	660000	1.20E-11	7.92E-06	6.39E-09	0.00422
Boucles FOR	**Int_0**	36000000	1.29E-10	4.65E-03	6.86E-08	2.46988
	Int_1	36000000	1.29E-10	4.65E-03	6.86E-08	2.46988
	Int_2	9600000	1.29E-10	1.24E-03	6.86E-08	0.65863
	Int_3	9600000	1.29E-10	1.24E-03	6.86E-08	0.65863
	Int_4	8400000	1.29E-10	1.09E-03	6.86E-08	0.5763
	Int_5	8400000	1.29E-10	1.09E-03	6.86E-08	0.5763
	Milieu	2400000	1.29E-10	3.10E-04	6.86E-08	0.16466
	Externe	20000	1.29E-10	2.58E-06	6.86E-08	0.00137
			Total:	1.97E-02	Total:	10.51719

Le tableau 7.1 se compose des divers éléments suivants :

1. **Instr.**: les instructions étudiées;
2. **Boucles FOR:** l'ensemble des boucles utilisées dans le programmes. Elles sont toutes identifiées dans le code de l'annexe II;
3. **Total instr.**: Le nombre total d'instructions exécutées par le programme de l'annexe II au travers de l'ensemble des boucles;
4. **Énergie Moyenne par instr.**: calculée pour les instructions au tableau 6.5. Donne l'énergie moyenne consommée par instruction;

5. **Énergie Totale par instr.**: **Total instr.** multiplié par **Énergie Moyenne par instr.**. Donne l'énergie totale consommée par type d'instructions;
6. **Temps Exécution Théorique**: calculé pour les instructions au tableau 6.1. Donne le temps d'exécution d'une seule instruction;
7. **Total Temps Exécution Théorique: Total instr.** multiplié par **Temps Exécution Théorique.** Donne le temps total que prends un type d'instruction pour s'exécuter.

Le tableau 7.1 montre que l'énergie théorique totale consommée par le programme est de 1.97E-02 Joules. Le temps d'exécution théorique total du programme est de 10.5 sec.

7.4 Caractéristiques énergétiques pratiques

Tableau 7.2 Caractéristiques énergétiques pratiques du filtre

Sortie Ampli		**Point de Test**					
Tension 15 cœurs Mesurée (V)	**Gain Ampli Mesuré**	**Tension 15 cœurs (Vrms)**	**Courant 15 cœurs (Irms)**	**Courant Moyen par cœur (Irms)**	**$T_{exéc}$ (cœur #1) (sec)**	**Tension Cœur Mesurée (Vrms)**	**Énergie Totale Dissipée (J)**
1.084	8670	1.25E-04	2.50E-02	1.67E-03	11.195	1.167	2.18E-02

Le tableau 7.2 se compose des divers éléments suivants :

1. **Sortie Ampli**: mesures prises à la sortie de l'amplificateur. Figure 2.3;
2. **Tension 15 cœurs Mesurée**: tension mesurée à la sortie de l'amplificateur de l'ensemble des 15 cœurs exécutant simultanément le même programme de l'annexe II;
3. **Gain Ampli Mesuré**: le gain de l'amplificateur mesuré en laboratoire. Ce gain est constant dans la gamme des tensions mesurées;
4. **Point de Test**: mesures prises de la résistance de 0.005 Ω sur la plaquette d'Octasic fournie par le manufacturier. Tous les courants des 15 cœurs passent par cette résistance. Figure 2.2;
5. **Tension 15 cœurs: Tension 15 cœurs Mesurée** divisé par le **Gain Ampli Mesuré.** Donne la tension des 15 cœurs au **Point de Test** aux bornes de la résistance de 0.005 Ω;

6. **Courant 15 cœurs**: **Tension 15 cœurs** divisée par la résistance de 0.005 **Ω.** Donne le courant total des 15 cœurs exécutant le programme;
7. $\mathbf{T_{exéc}}$: Le temps mesuré que met le programme pour s'exécuter. Cette mesure a été prise pour un seul cœur (le cœur #1);
8. **Tension Cœur Mesurée**: tension alimentant les cœurs, mesurée en laboratoire. Les spécifications du manufacturier indiquent: Min 1.11 V, TYP 1.16 V, MAX 1.21 V;
9. **Énergie Totale Dissipée**: **Courant Moyen par cœur** multiplié par **Tension Cœur Mesuré** multiplié par $\mathbf{T_{exéc}}$. Donne l'énergie moyenne consommée pour chaque cœur.

Le tableau 7.2 illustre les caractéristiques énergétiques mesurées de notre filtre « FIR ». L'énergie totale consommée par le programme est de 2.18E-02 Joules et le temps d'exécution total du programme est de 11.2 sec.

7.5 Conclusion

Le tableau 7.1, donne l'énergie théorique totale consommée par le programme de 1.97E-02 Joules. Le tableau 7.2 donne l'énergie totale mesurée par le programme de 2.18E-02 Joules. Il en découle une erreur relative de 9.6%.

Le tableau 7.1, illustre également le temps d'exécution théorique total du programme de 10.5 sec. Le tableau 7.2 donne un temps d'exécution total mesuré du programme de 11.2 sec. Il en découle une erreur relative de 6.3%.

Ces différences de 9.6% et 6.3% s'expliquent par les raisons suivantes :

1. Toutes les mesures théoriques de temps d'exécution et de consommation d'énergie ont été faites alors qu'il n'y avait aucune dépendance de variables entre les instructions étudiées. Ce n'est pas le cas avec le filtre « FIR »;

2. Toutes les mesures théoriques de temps d'exécution et de consommation d'énergie ont été faites alors que le programme théorique étudié était dans sa phase linéaire. Cela implique des répétitions successives importantes. Ce n'est pas toujours le cas avec le filtre « FIR ».

Malgré cela, nous croyons que notre modèle reste une très bonne approximation énergétique de la réalité. Plus de précision amènerait une plus grande complexité du modèle.

CHAPITRE 8

INTERMODULATION DES COEURS

8.1 Introduction

Ce chapitre présente une image de l'ensemble du spectre des fréquences associé au fonctionnement du processeur asynchrone à quinze cœurs exécutant un programme en boucle. Le spectre de fréquences contient énormément de fréquences produites par ce que nous supposons être de l'intermodulation des cœurs entre eux. Ces travaux se limitent à l'intermodulation du second ordre tel que décrit dans le manuel d'utilisation du Spectrum Anritsu Intermodulation Distortion (IMD) Measurements Using the 37300 Series Vector Network Analyser [1]. Mentionnons que l'intermodulation du second ordre consiste en l'observation des fréquences suivantes : f_1+f_2, f_1-f_2, $2f_1$, $2f_2$, f_1 et f_2. Suite à cela, la vitesse d'exécution de chacun des cœurs est estimée et présentée. Le but de ces travaux exploratoires est d'investiguer le potentiel du phénomène d'intermodulation dans la caractérisation des processeurs asynchrones.

8.2 Méthodologie

La méthodologie consiste à exécuter un programme en boucle utilisant la même instruction assez longtemps pour en voir les effets d'intermodulation à l'analyseur de spectres. Le compteur cadencé est utilisé de la même manière qu'à la section 3.2. La figure 8.1 présente un exemple de code utilisé. Il est à remarquer que le code de la figure 8.1 ne contient aucune dépendance entre les variables.

```
_asm rdtsc timerStart;

register unsigned int Val1 = 1;
register unsigned int Val2 = 1;
register unsigned int Val3 = 0;

for(i=0; i<600; i++)
    {
        for(j=0; j<65536; j++)
          {
             _asm mac.ll Val3, Val1, Val2; //#inst. varie
          }
    }

_asm
    {
        rdtsc timerStop;
        sub timerDiff, timerStart, timerStop;
    }
```

Figure 8.1 Exemple de code produisant de l'intermodulation

8.3 Présentation globale du spectre de fréquences d'intermodulation

Le tableau 8.1 présente une vue d'ensemble du spectre de fréquences du VEB lorsque le programme de la figure 8.1 est exécuté.

Tableau 8.1 Spectre de fréquences d'intermodulation

Item	Fréquences	Commentaires
1.	0 @ 2 MHz	Fréquences d'intermodulation et harmoniques principalement produites par l'item #8 ($f_1 - f_2$)
2.	4 @ 7 MHz	Fréquences d'intermodulation et harmoniques produites par l'item #7 ($f_1 - f_2$)
3.	8 @ 13 MHz	Fréquences d'intermodulation et harmoniques produites par l'item #2 ($f_1 + f_2$, $f_1 - f_2$)
4.	60 @ 100 MHz	Demande plus d'analyse
5.	150 MHz	Fréquence isolée
6.	200 MHz	Fréquences isolées
7.	300 @ 500 MHz	Enveloppe des « ALU »
8.	Région 865 MHz	Fréquence des cœurs
9.	1.96 GHz	$5^{ième}$ harmonique « ALU »
10.	2.4 GHz	$6^{ième}$ harmonique « ALU »

Toutes les fréquences observées au tableau 8.1 sont le fruit du programme de la figure 8.1 exécutant cinquante instructions MAC à l'intérieur de 2 boucles imbriquées. Tous les quinze cœurs fonctionnent en même temps de façon parallèle en exécutant le même programme.

8.4 Présentation détaillée du spectre de fréquences d'intermodulation

Cette section présente en détail chacun des éléments du tableau 8.1. Dans chacun des cas, une image du spectre étudié directement prise de l'analyseur de spectres est présentée. Rappelons que compte tenu des spécifications du manufacturier et du produit gain, bande passante de l'amplificateur opérationnel utilisé, les niveaux exacts de puissance des différentes fréquences des captures d'écran de l'analyseur de spectres, ne sont pas pertinents. Par contre, l'apparition des composantes en fréquences lors du fonctionnement du programme indique bel et bien des produits d'intermodulation générés par la demande de courant du processeur.

8.4.1 Fréquences d'intermodulation des cœurs

La figure 8.2 montre une capture d'écran de l'analyseur de spectres de l'item 1 du tableau 8.1, des fréquences de 0 @ 2 MHz.

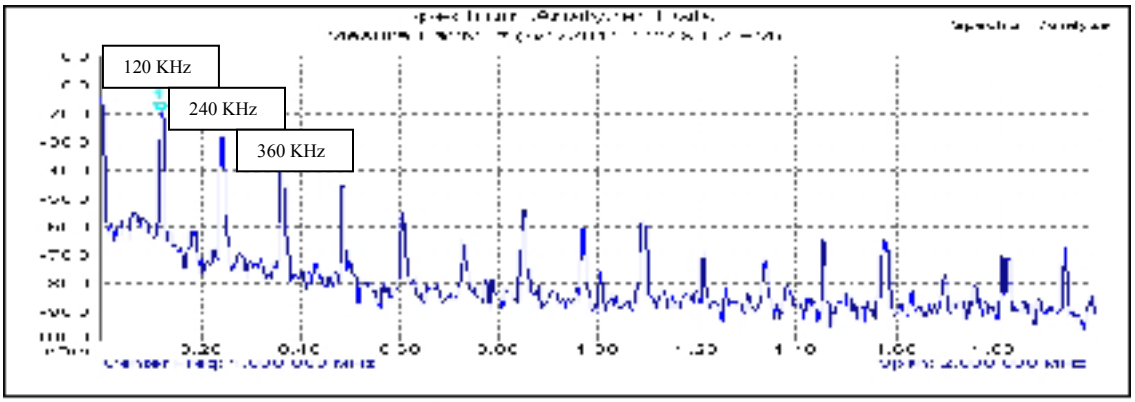


Figure 8.2 Spectre d'intermodulation des cœurs

Plus d'une douzaine d'harmoniques de la fréquence de 120 KHz sont observées. Nous considérons que cette fréquence constitue la fréquence d'intermodulation d'ordre 2, suite à l'investigation décrite dans ce qui suit.

Tableau 8.2 Fréquences des cœurs

Item	**Fréquences des cœurs (MHz)**	**Δ avec l'item précédent (KHz)**
1	864.492726	
2	864.618181	125.455
3	864.732726	114.545
4	864.852726	120.000
5	864.972726	120.000
6	865.098181	125.455
7	865.223636	125.455
8	865.338181	114.545
9	865.458181	120.000
10	865.572726	114.545
11	865.692726	120.000
12	865.818181	125.455
13	865.938181	120.000
14	866.058181	120.000
15	866.172726	114.545

Le tableau 8.2 montre la fréquence des cœurs, présentée dans un ordre croissant, et la différence de fréquences entre chaque cœur adjacent dans la liste. Cette différence représente la fréquence d'intermodulation des cœurs. Les quinze fréquences de la deuxième colonne (autour de 865 MHz) sont associées aux quinze cœurs du processeur car la capture d'écran faite à ces fréquences montre quinze raies distinctes. Cette capture d'écran de l'analyseur de spectres est présentée plus loin à la figure 8.9. La fréquence moyenne de la colonne **Δ avec l'item précédent**, du tableau 8.2 est exactement de 120.000 KHz. Cette fréquence représente la différence fréquentielle des raies de la figure 8.2. C'est pourquoi il est considéré que ces raies sont le fruit des produits d'intermodulation entre les fréquences des cœurs. Par conséquent, l'analyse de ces fréquences offre un certain potentiel au niveau de la caractérisation, pour la mesure à plus basse fréquence, de la différence des cœurs asynchrones.

8.4.2 Fréquences d'intermodulation d'enveloppe des « ALU »

La figure 8.3 montre une capture d'écran de l'analyseur de spectres de l'item 2 du tableau 8.1, des fréquences de 4 @ 7 MHz.

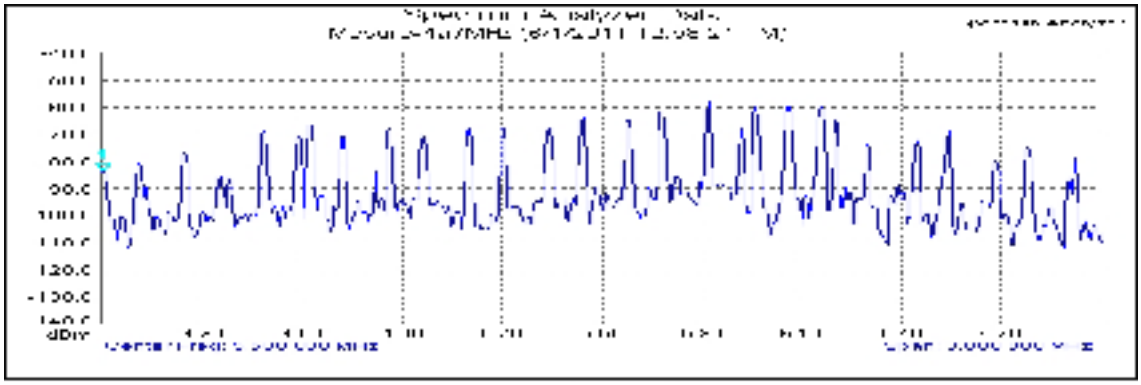


Figure 8.3 Spectre d'intermodulation des « ALU »

Plus d'une vingtaine de fréquences sont observées. Une vérification minutieuse de ces fréquences permet de conclure qu'il s'agit des fréquences d'intermodulation ($f_1 - f_2$) produites par les fréquences de l'item #7. Leur grand nombre rend plus difficile leur utilisation dans un processus de caractérisation. Il pourrait par contre être pertinent d'explorer cette plage de fréquence avec un seul cœur en opération (travaux futurs).

8.4.3 Spectre des fréquences de 8 @ 13 MHz

La figure 8.4 présente une capture d'écran de l'analyseur de spectres de l'item 3 du tableau 8.1, des fréquences de 8 @ 13 MHz. Une vérification minutieuse de ces fréquences nous permet de conclure qu'il s'agit des produits d'intermodulation ($f_1 + f_2$, $f_1 - f_2$) des fréquences présentées dans la section 8.4.2. Ces fréquences sont extrêmement dynamiques et dont l'amplitude varie sans arrêt. Encore ici, leur grand nombre rend plus difficile leur utilisation dans un processus de caractérisation.

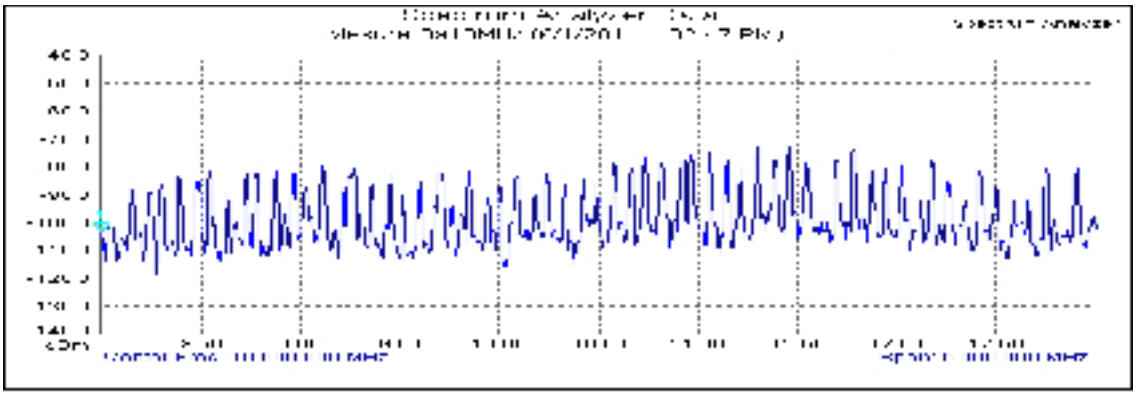


Figure 8.4 Spectre d'intermodulation produit par les fréquences d'intermodulation des enveloppes des « ALU »

8.4.4 Spectre des fréquences de 60 @ 100 MHz

La figure 8.5 présente une capture d'écran de l'analyseur de spectres de l'item 4 du tableau 8.1, des fréquences de 60 @ 100 MHz.

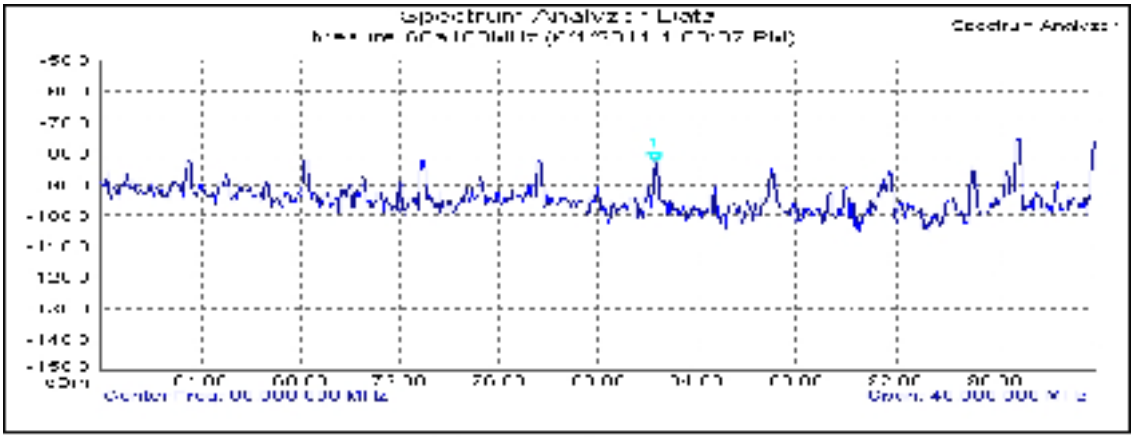


Figure 8.5 Spectre des fréquences de 60 @ 100 MHz

Bien que ces fréquences n'affichent pas de grandes amplitudes, elles sont très distinctes sur l'écran de l'analyseur de spectres. Cet ensemble de fréquences demande une analyse plus complète afin d'en déterminer la provenance. Ces travaux débordent du cadre de ce mémoire.

8.4.5 Fréquence isolée de 150 MHz

La figure 8.6 présente une capture d'écran de l'analyseur de spectres de l'item 5 du tableau 8.1, d'une fréquence unique de 150 MHz qui est l'une des horloges de la carte du VBE.

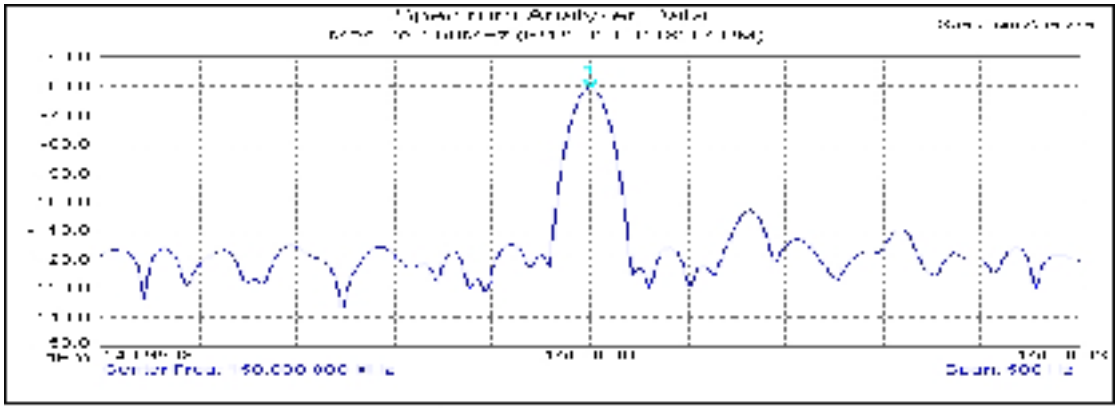


Figure 8.6 Spectre des fréquences de 150 MHz

8.4.6 Fréquences isolées 200 MHz

La figure 8.7 présente une capture d'écran de l'analyseur de spectres de l'item 6 du tableau 8.1, de deux fréquences isolées aux alentours de 200 MHz. La provenance de ces fréquences demande une analyse plus approfondie (travaux futurs).

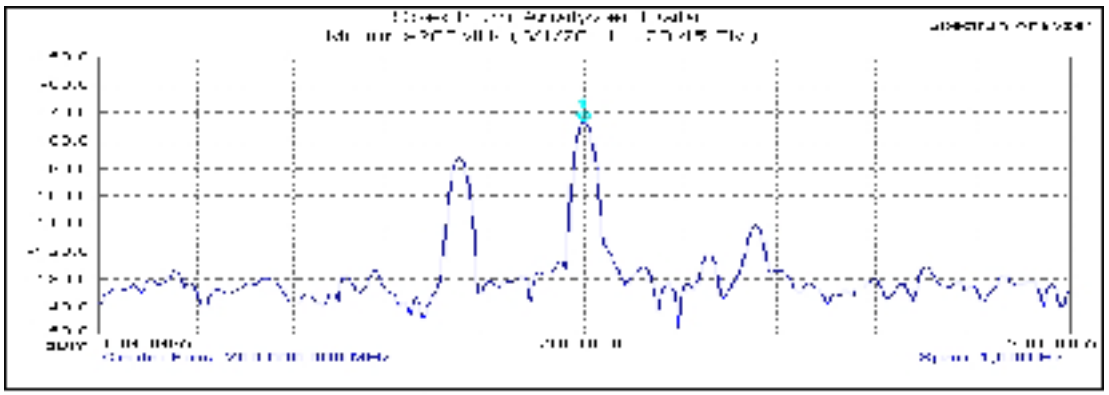


Figure 8.7 Spectre des fréquences 200 MHz

8.4.7 Spectre des fréquences d'enveloppes des « ALU »

La figure 8.8 présente une capture d'écran de l'analyseur de spectres de l'item 7 du tableau 8.1, des fréquences de 300 @ 500 MHz. En raison de la valeur des délais combinatoire des « ALU », nous croyons que plusieurs de ces fréquences sont produites par l'exécution de l'instruction MAC. Cette hypothèse est appuyée par les mesures de fréquences des instructions par cœur (tableau 8.3, section 8.5). C'est pourquoi nous les appelons fréquences d'enveloppes des « ALU ». Ces fréquences sont extrêmement dynamiques et dont l'amplitude varie sans arrêt.

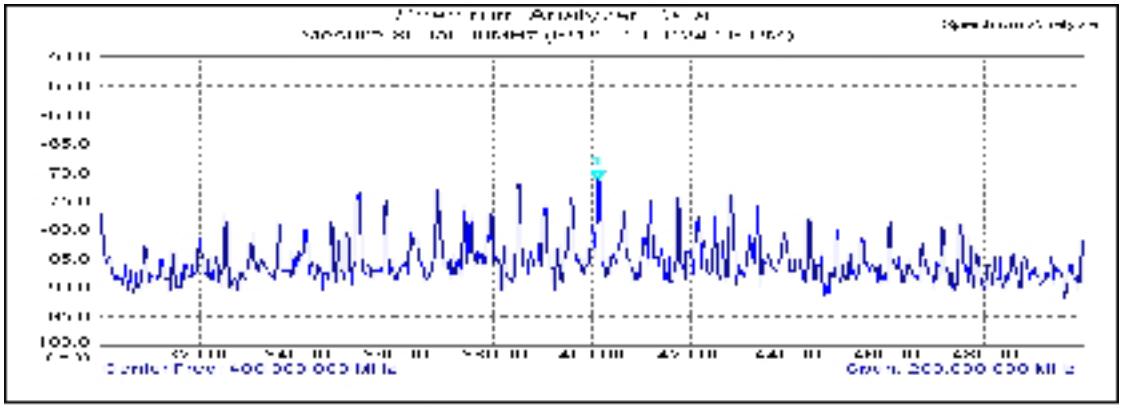


Figure 8.8 Spectre des fréquences d'enveloppe des « ALU »

8.4.8 Spectre de fréquences des cœurs

La figure 8.9 montre une capture d'écran de l'analyseur de spectres de l'item 8 du tableau 8.1, des fréquences aux environs de 865 MHz.

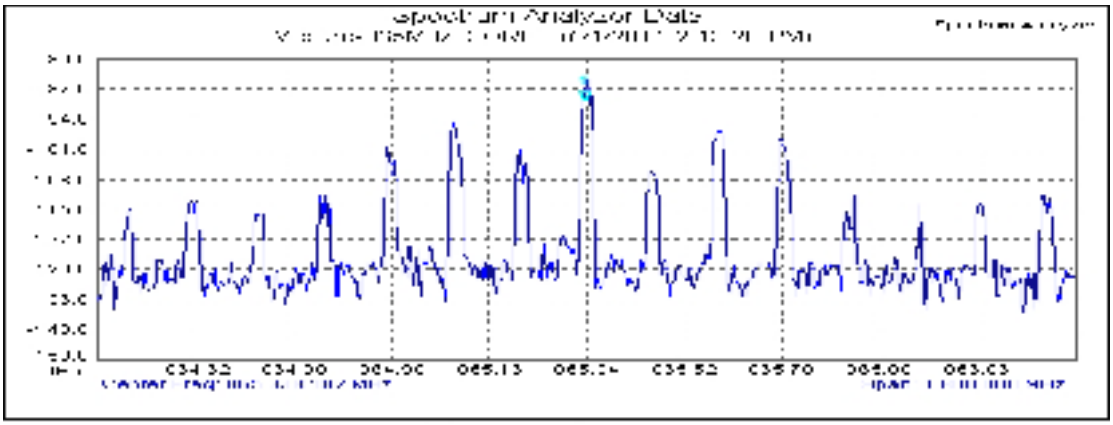


Figure 8.9 Spectre des fréquences des cœurs

La figure 8.9 présente le spectre des fréquences d'exécution des cœurs. Ces fréquences sont très stables et distinctes. Comme il a été mentionné précédemment, nous associons ces 15 fréquences aux quinze cœurs du processeur.

8.4.9 Spectre de fréquences 5ième harmonique d'enveloppe des « ALU »

La figure 8.10 présente une capture d'écran de l'analyseur de spectres de l'item 9 du tableau 8.1 des fréquences aux environs de 1.96 GHz.

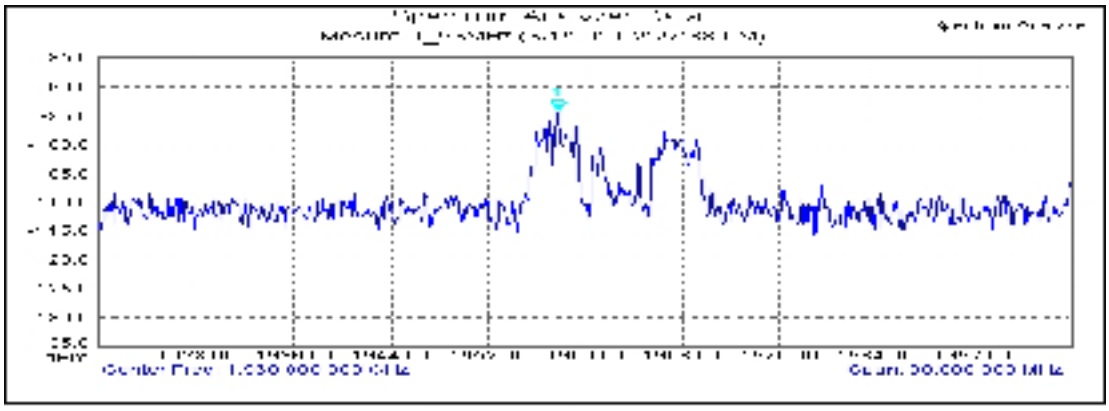


Figure 8.10 Spectre des fréquences 5ième harmonique d'enveloppe des « ALU »

La figure 8.10 présente le spectre de fréquences de ce qui semble être la 5ième harmonique d'enveloppe des « ALU ». Rappelons que le spectre de fréquences d'enveloppes des « ALU

» se situe aux environs de 300 @ 500 MHz. Des investigations supplémentaires sont cependant nécessaires afin de valider la provenance et de statuer sur l'utilité de ce spectre dans le processus de caractérisation.

8.4.10 Spectre de fréquences 6ième harmonique d'enveloppe des « ALU »

La figure 8.11 présente une capture d'écran de l'analyseur de spectres de l'item 10 du tableau 8.1 des fréquences aux environs de 2.4 GHz.

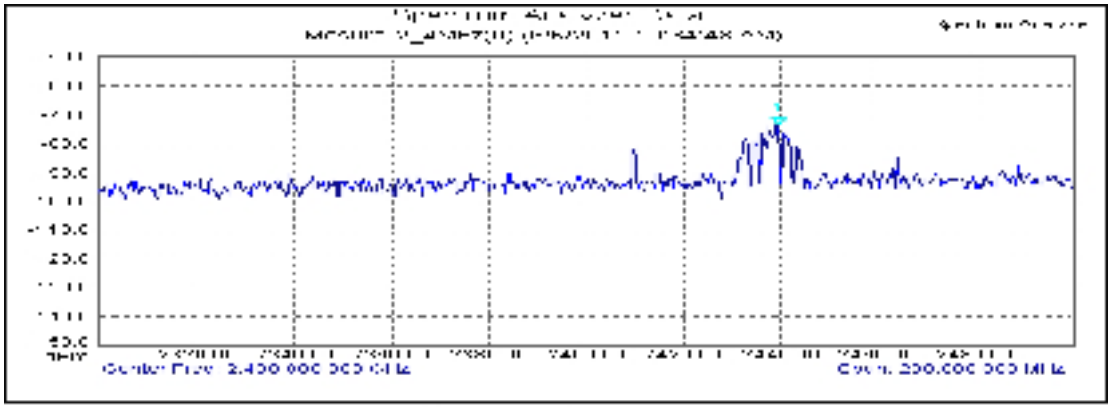


Figure 8.11 Spectre des fréquences 6ième harmonique d'enveloppe des « ALU »

La figure 8.11 présente le spectre de fréquences de ce qui semble être la 6ième harmonique d'enveloppes des « ALU ». Des investigations supplémentaires sont également ici nécessaires afin de valider la provenance et de statuer sur l'utilité de ce spectre dans le processus de caractérisation.

8.5 Vitesse d'exécution des cœurs

Cette section se concentre sur la mesure de la vitesse d'exécution des quinze cœurs du VEB. Le programme utilisé à cette fin est présenté à la figure 8.1. Le tableau 8.3 illustre les mesures prises.

Tableau 8.3 Vitesse d'exécution des cœurs

Item	Cœur	Vitesse d'exécution des cœurs en MHZ par instruction				
		50 X MAC	25 X MAC	10 X MAC	5 X MAC	Boucle vide
1.	#1	357.560	393.518	505.891	689.939	907.583
2.	#2	359.959	397.227	510.473	697.065	919.631
3.	#3	360.256	397.848	511.370	698.257	919.864
4.	#4	358.106	395.554	508.689	694.748	915.378
5.	#5	361.242	399.169	513.356	700.914	923.141
6.	#6	360.436	398.176	511.867	699.285	918.663
7.	#7	358.004	395.515	508.590	694.770	918.966
8.	#8	362.246	400.282	514.556	702.800	925.296
9.	#9	360.246	397.897	511.612	698.557	918.377
10.	#10	357.438	394.799	507.808	693.091	910.820
11.	#11	359.505	396.897	510.672	696.814	916.908
12.	#12	357.583	394.834	507.931	693.399	913.069
13.	#13	355.406	398.181	504.850	688.799	905.748
14.	#14	358.910	396.389	509.907	695.460	915.432
15.	#15	355.448	392.610	504.875	688.960	905.410

Rappelons qu'un programme de deux boucles imbriquées contenant les instructions « MAC » est utilisé. La boucle externe (for (i=0; i<600; i++)) introduit 4 instructions en langage machine, générées par le compilateur. La boucle interne (for (j=0; i<65536; j++)) introduit 6 instructions en language machine, générées par le compilateur.

Afin de prendre des mesures significatives, le nombre d'itérations de la boucle externe peut varier. La formule suivante a été utilisée afin de calculer le nombre total d'instructions exécutées à l'intérieur des deux boucles imbriquées étudiées :

$$Total\ des\ instructions = [(B_{ext} * B_{int} * (N_{mac} + 6] + [B_{ext} * 4] \qquad (8.1)$$

Le compteur cadencé est utilisé afin de comptabiliser le temps d'exécution du programme de la figure 8.1 pour chaque groupe d'instructions « MAC » étudié du tableau 8.3. En divisant le nombre total d'instructions étudiées grâce à l'équation 8.1, par le temps d'exécution du programme, cela donne un temps d'exécution par instruction. L'inverse de ce temps donne la fréquence d'exécution par instruction que l'on retrouve au tableau 8.3. Le tableau 8.4 fait ressortir les points importants du tableau 8.3.

Tableau 8.4 Vitesse d'exécution des cœurs - Sommaire

Item	**Cœur**	**50 X MAC** (MHz par instruction)	**25 X MAC** (MHz par instruction)	**10 X MAC** (MHz par instruction)	**5 X MAC** (MHz par instruction)	**Boucle vide** (MHz par instruction)
1.	Plus rapide (# cœur)	362.246 (#8)	400.282 (#8)	514.556 (#8)	702.800 (#8)	925.296 (#8)
2.	Plus lent (# cœur)	355.406 (#13)	392.610 (#15)	504.850 (#13)	688.960 (#15)	905.410 (#15)
3.	Moyenne	358.826	396.446	509.716	695.880	915.353
4.	Écart	6.840	7.672	9.681	13.840	19.886

Le tableau 8.4 fait ressortir le fait que le cœur treize et le cœur quinze se partagent la place du cœur le plus lent, tandis que le cœur 8 est le cœur le plus rapide. Le tableau 8.3 fait également ressortir le fait que chacun des cœurs à sa vitesse d'exécution propre tel que décrit par le spectre de fréquences des cœurs de la section 8.4.8.

8.6 Conclusion

L'objet de ce chapitre était de présenter, dans un premier temps, une image de l'ensemble du spectre de fréquences émis par un processeur asynchrone à quinze cœurs exécutant un programme en boucle et d'explorer leur potentiel dans un processus de caractérisation du processeur. Le spectre contient énormément de fréquences. Les composantes principales se situent autour de 865 MHz. Ces dernières sont très stables et distinctes, et ont été associées à la fréquence d'exécution des cœurs. Un deuxième ensemble de fréquences d'intérêt se situe dans la plage de 0 à 2 MHz. L'analyse de ces fréquences a permis de conclure qu'elles

correspondaient aux produits d'intermodulation des cœurs entre eux. Ces deux plages de fréquences présentent un potentiel intéressant de caractérisation, en particulier pour ce qui est de mesurer la différence, absolue ou relative, de vitesse de fonctionnement des cœurs asynchrones. Les autres fréquences observées dans le spectre présentent moins d'intérêt.

Dans un deuxième temps, la vitesse d'exécution des différents cœurs par instruction a été estimée et il en ressort que chacun des cœurs à sa vitesse propre.

CONCLUSION

Le premier chapitre consiste en une revue des connaissances du sujet traité et positionne les présents travaux. Le second chapitre porte sur l'environnement de test utilisé dans le cadre de ce projet, y compris la carte VEB sur laquelle les travaux sont effectués.

Au troisième chapitre, un programme simple à 2 boucles imbriquées a été paramétré. Ceci impliquait la paramétrisation de la boucle externe, de la boucle interne et des instructions contenu dans la boucle interne du programme. Dans le cas de la paramétrisation de N_{mac}, il a été démontré que $T_{exéc}$ est linéaire pour $N_{mac} > 5$. Dans le cas des paramétrisations de B_{int} et B_{ext}, il a été démontré que $T_{exéc}$ est entièrement linéaire. Ceci suggère que les coûts fixes en termes de temps d'exécution deviennent alors négligeables.

Au chapitre 4, la modélisation avec des équations mathématiques de la paramétrisation de notre programme à 2 boucles imbriquées est présentée. Cette modélisation nous amène à calculer une vitesse d'exécution distincte ainsi qu'un temps d'exécution propre à chaque instruction pour un processeur asynchrone. La précision de notre modèle a été établie en comparant le temps d'exécution mesuré au temps d'exécution théorique calculé. Dans chacun des cas, une excellente précision avec un grand nombre d'itérations ou d'instructions a été obtenue.

Le cinquième chapitre montre qu'il est possible de produire des fréquences d'exécution d'instructions et de boucles en exécutant un programme en boucle. La période de ces fréquences d'exécution représente le temps d'exécution propre de ces instructions pour un processeur asynchrone. L'hypothèse selon laquelle un programme fonctionnant en boucle provoque une demande de courant à la source d'alimentation à chaque fois qu'une instruction est exécutée et que cette répétition va se traduire par l'apparition, dans le spectre fréquentiel de la consommation de courant, de la fréquence d'exécution de l'instruction et des boucles internes et externes est donc validée. Concernant la boucle externe de notre programme,

notre modèle indique une fréquence spécifique d'exécution alors que l'expérience reproduit sa deuxième harmonique. À ce jour, nous n'avons pas d'explication à ce phénomène. Plusieurs hypothèses sont envisagées, dont l'influence des mécanismes internes de transfert de données dans le contexte de l'insuffisance de ressources internes des cœurs mais la validation de celles-ci dépasse le cadre de ce travail.

Au chapitre 6, le temps propre d'exécution des instructions typiques servant à former un programme pour filtre FIR est calculé. La consommation moyenne en puissance de chaque cœur pour chaque type d'instructions est estimée. La consommation énergétique moyenne de chaque instruction est également estimée. Du temps d'exécution des instructions, il est possible de calculer le temps d'exécution théorique total de chaque programme. Enfin, l'erreur relative entre le temps calculé d'exécution des programmes et leur temps réel mesuré a été montrée. Ces erreurs relatives sont faibles pour toutes les instructions en assembleur, ce qui confirme la validité du modèle mathématique du chapitre 4.

Le chapitre 7 montre l'application de notre modèle à un filtre réel de type « FIR ». L'énergie théorique totale consommée par le programme est de 1.97E-02 Joules, alors que l'énergie totale mesurée par le programme est de 2.18E-02 Joules. Une erreur relative de 9.6% les sépare. D'autre part, le temps d'exécution théorique total du programme est de 10.5 sec alors que le temps d'exécution total mesuré du programme de 11.2 sec. Une erreur relative de 6.1% les sépare.

Ces différences de 9.6% et 6.3% s'expliquent par les raisons suivantes :

1. Toutes les mesures théoriques de temps d'exécution et de consommation d'énergie ont été faites alors qu'il n'y avait aucune dépendance de variables entre les instructions étudiées. Ce n'est pas le cas avec le filtre « FIR »;
2. Toutes les mesures théoriques de temps d'exécution et de consommation d'énergie ont été faites alors que le programme théorique étudié était dans sa phase linéaire. Ceci implique des répétitions successives importantes. Ce n'est pas toujours le cas avec le filtre « FIR ».

Malgré cela, nous croyons que notre modèle reste une très bonne approximation énergétique de la réalité. Plus de précision amènerait une plus grande complexité du modèle.

Le dernier chapitre présente, dans un premier temps, une image de l'ensemble du spectre de fréquences émis par un processeur asynchrone à quinze cœurs exécutant un programme en boucle et explore leur potentiel dans un processus de caractérisation du processeur. Le spectre contient énormément de fréquences. Les composantes principales se situent autour de 865 MHz. Ces dernières sont très stables et distinctes, et ont été associées à la fréquence d'exécution des cœurs. Un deuxième ensemble de fréquences d'intérêt se situe dans la plage de 0 à 2 MHz. L'analyse de ces fréquences a permis de conclure qu'elles correspondaient aux produits d'intermodulation des cœurs entre eux. Ces deux plages de fréquences présentent un potentiel intéressant de caractérisation, en particulier pour ce qui est de mesurer la différence, absolue ou relative, de vitesse de fonctionnement des cœurs asynchrones. Les autres fréquences observées dans le spectre présentent moins d'intérêt. Dans un deuxième temps, la vitesse d'exécution des différents cœurs par instruction a été estimée et il en ressort que chacun des cœurs à sa vitesse propre.

Dans l'ensemble, les présents travaux proposent une technique distincte d'estimation de consommation de puissance et d'énergie pour des programmes logiciels codés en assembleur et exécutés par un processeur multicoeur asynchrone en traitement parallèle. Cette technique se veut simple et de première approximation. Cette technique, de même que les efforts de caractérisation poussés du processeur multicoeur asynchrone, constituent les principales contributions de ce mémoire.

Rappelons que les principaux efforts de caractérisations du processeur multicoeur asynchrone, comprennent l'estimation de la vitesse d'exécution des instructions, l'estimation de la vitesse d'exécution de chaque cœur ainsi que les produits d'intermodulation des fréquences d'exécution des cœurs.

RECOMMANDATIONS

Les présents travaux se sont limités à caractériser en puissance les quelques instructions servant à la programmation d'un filtre «FIR» typique. Il pourrait être souhaitable de caractériser l'ensemble de toutes les instructions en assembleur du processeur asynchrone afin d'avoir une bibliothèque complète.

Dans un deuxième temps, les bibliothèques de caractérisation en puissance et en énergie développées dans le cadre de ces travaux, pourraient être utilisées dans les recherches ultérieures faisant partie des phases subséquentes du projet OPÉRA. Le développement d'algorithmes logiciels le moins énergivores possibles reste une des pistes de solutions au problème croissant de la demande énergétique des puces VLSI.

De plus, il serait intéressant d'appliquer les techniques utilisées dans ces travaux pour les applications de tests d'interférences électromagnétiques.

ANNEXE I

EXEMPLE DE PROGRAMME UTILISÉ POUR CARACTÉRISER LES INSTRUCTIONS

```
// Project name: Charge_DSP
#include <dma_util.h>
int main()
{
	register unsigned int timerStart, timerStop, timerDiff;
	register unsigned int i, j, k;
	register unsigned int Val1 = 0;
	register unsigned int Val2 = 1;
	if (get_core_number() == 1)
		{
		}
	else
		{
			return 0;
		}

	_asm rdtsc timerStart;

	for(i=0; i<256 i++)                                    //Boucle de charge
		{
			for(j=0; j<65536; j++)
				{
				  _asm mov Val1, Val2;        //Instruction étudiée
				}
		}
	_asm
		{
			rdtsc timerStop;
			sub timerDiff, timerStart, timerStop;
			nop;
		}
	return 0;
}
```

ANNEXE II

EXEMPLE DE PROGRAMME UTILISÉ POUR CARACTÉRISER UN FILTRE FIR

```
// Project name: FIR_FINAL_DSPIC
// main_FIR_FINAL_DSPIC.c  **CECI EST LA VERSION FINALE DU PROJET FILTRE
//POUR //VOCALLO VERSION HARDWARE
// AVEC UNE TABLE DE SINUS**

//Note: Tous les tests concordent a 2 chiffres apres le point avec le programme
//FIR_Decimation_Table_Sinus(ecrit en C)
//
// 1.     // 2.     Filtre FIR - Direct Form - Decimation - F etudiees = 8KHz (120 echantillons)
et 16KHz (60 echantillons)
//
//                Decimation par 5
//                        a) Fpass = 10KHz
//                        b) Fstop = 180KHz
//                        c) Fs     = 960KHz
//                        d) Ordre = 15
//                        e) Coefficients = 16
//
//                Decimation par 4
//                        a) Fpass = 10KHz
//                        b) Fstop = 36KHz
//                        c) Fs     = 192KHz
//                        d) Ordre = 20
//                        e) Coefficients = 21
//
//                Decimation par 2
//                        a) Fpass = 10KHz
//                        b) Fstop = 12KHz
//                        c) Fs     = 48KHz
//                        d) Ordre = 70
//                        e) Coefficients = 71

//        Tous les signaux de sortie decimes sont lus avec Fs = 24KHz
//
// AUTHEUR:   Francois Gourlay
// DATE: 11-OCT-2010
// ORGANISATION:  LACIME, Ecole de technologie superieure de Montreal
//
// Historique des versions
// ----------------------
```

```
// 1.0     11-OCT-2010 Version Initiale
// 1.1
//
//
#include <dma_util.h>
#include "Coeff_Decimation.h"
int main(void)
{
        if (get_core_number() == 1)
                {
                }
        else if (get_core_number() == 2)
                {
                }
        else if (get_core_number() == 3)
                {
                }
        else if (get_core_number() == 4)
                {
                }
        else if (get_core_number() == 5)
                {
                }
        else if (get_core_number() == 6)
                {
                }
        else if (get_core_number() == 7)
                {
                }
        else if (get_core_number() == 8)
                {
                }
        else if (get_core_number() == 9)
                {
                }
        else if (get_core_number() == 10)
                {
                }
        else if (get_core_number() == 11)
                {
                }
        else if (get_core_number() == 12)
                {
                }
        else if (get_core_number() == 13)
```

```
		{
		}
	else if (get_core_number() == 14)
		{
		}
	else if (get_core_number() == 15)
		{
		}
	else
		{
		return 0;
		}
	register unsigned int timerStart;
	register unsigned int timerStop;
	register unsigned int timerDiff;
	int i;
	int j;
	int k;
	int p;



//****GENERATION DU FILTRE FIR en format Q ************


	register signed __int32 count_cycle = 15000;
	register signed __int32 Periode = 120;
	register signed __int32 Ordre_FIR_Deci5 = 15;
	register signed __int32 Xn_Deci5_Q[16];
	register signed __int32 Ordre_FIR_Deci4 = 20;
	register signed __int32 Xn_Deci4_Q[21];
	register signed __int32 Ordre_FIR_Deci2 = 70;
	register signed __int32 Xn_Deci2_Q[71];
	register signed __int32 Yn_Fin_Q[3];
	for (i=0; i<Ordre_FIR_Deci5+1; i++)
	//Initialisation de la table des entres @ 0
	{
		Xn_Deci5_Q[i] = 0;
	}
	for (i=0; i<Ordre_FIR_Deci4+1; i++)
	//Initialisation de la table des entres @ 0
	{
		Xn_Deci4_Q[i] = 0;
	}
```

```
        for (i=0; i<Ordre_FIR_Deci2+1; i++)
        //Initialisation de la table des entres @ 0
        {
                Xn_Deci2_Q[i] = 0;
        }
//*************************************************************************
//Début du compteur de cycles avant la boucle du filtre
//*************************************************************************
        _asm rdtsc timerStart;
        //Compteur de cycles

        for (k=0; k<count_cycle; k++)           //Boucle Externe
                                                //Nombre de cycles de Sinus a lire
        {

                for (i=0; i<Periode; i++)       //Boucle Milieu
                                                //Lecture du Sinus
                {

//Decimation par 5

register signed __int32 Temp_Xn_Deci5_Q = 0;
register signed __int32 Temp_Bn_Deci5_Q = 0;
register signed __int32 Temp_YnTemp_Deci5_Q = 0;
register signed __int32 Temp_Yn_Fin_Deci5_Q = 0;
register signed __int32 Temp_Deci5 = 0;
register signed __int32 Temp_Xn_Deci4_Q = 0;
register signed __int32 Temp_Bn_Deci4_Q = 0;
register signed __int32 Temp_YnTemp_Deci4_Q = 0;
register signed __int32 Temp_Yn_Fin_Deci4_Q = 0;
register signed __int32 Temp_Deci4 = 0;
register signed __int32 Temp_Xn_Deci2_Q = 0;
register signed __int32 Temp_Bn_Deci2_Q = 0;
register signed __int32 Temp_YnTemp_Deci2_Q = 0;
register signed __int32 Temp_Yn_Fin_Deci2_Q = 0;
register signed __int32 Temp_Deci2 = 0;
Xn_Deci5_Q[0] = Sin_Q[i];
for (j=0; j<Ordre_FIR_Deci5; j++)    //Boucle Int_0
//Calcul des sorties du filtre
        {
                Temp_Xn_Deci5_Q = Xn_Deci5_Q[j];
                Temp_Bn_Deci5_Q = Bn_Deci5_Q[j];
                _asm        mac.ll        Temp_YnTemp_Deci5_Q,        Temp_Xn_Deci5_Q,
                Temp_Bn_Deci5_Q;
```

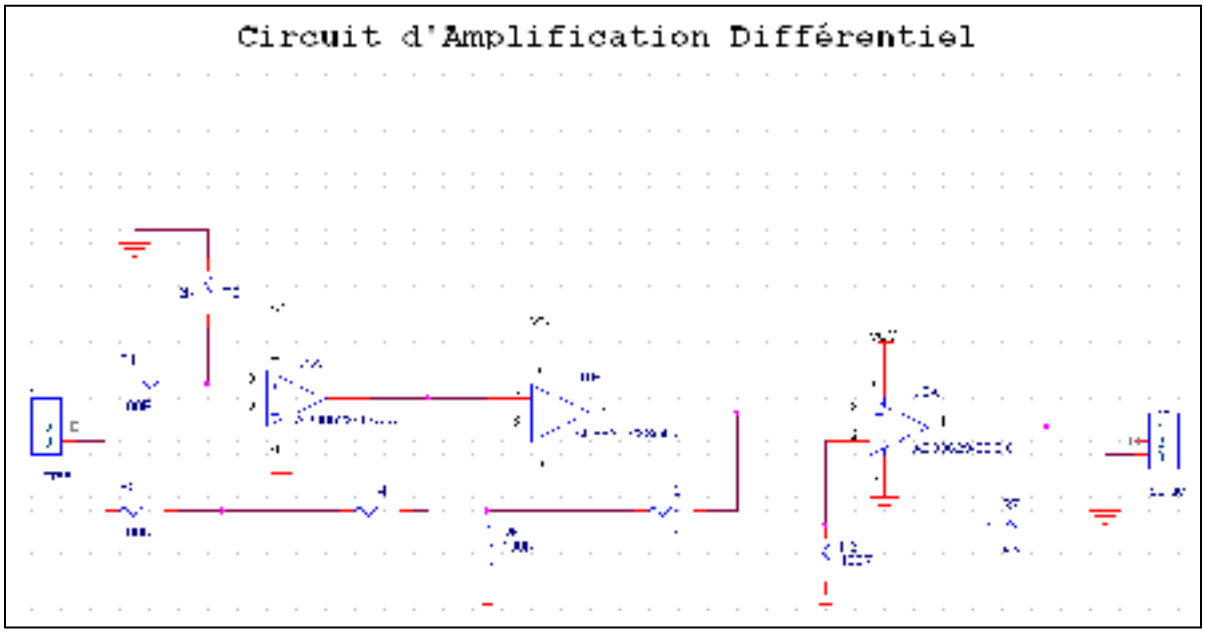
```
}
_asm mov Temp_Deci5, 1;              //Retourne du format Q2.28 (apres 2 multiplications)
_asm shl Temp_Yn_Fin_Deci5_Q, Temp_Deci5,   Temp_YnTemp_Deci5_Q;
_asm mov Temp_Deci5, 16;
_asm shr Temp_Yn_Fin_Deci5_Q, Temp_Deci5, Temp_Yn_Fin_Deci5_Q;
_asm nop;

//*************************************************************************
for(j=0; j<Ordre_FIR_Deci5; j++)     //Boucle Int_1
                                     //Decalage temporel des entrees Xn
{
Xn_Deci5_Q[Ordre_FIR_Deci5-j] = Xn_Deci5_Q[Ordre_FIR_Deci5-1-j];
}

_asm nop;

//Decimation par 4*********************************************************

if (i % 5 == 0)                      //IF_0
                                     //Decimation par 5
        {
                Xn_Deci4_Q[0] = Temp_Yn_Fin_Deci5_Q;
                for (j=0; j<Ordre_FIR_Deci4; j++)    //Boucle Int_2
                                                     //Calcul des sorties du filtre
                        {
                                Temp_Xn_Deci4_Q = Xn_Deci4_Q[j];
                                Temp_Bn_Deci4_Q = Bn_Deci4_Q[j];
                                _asm  mac.ll  Temp_YnTemp_Deci4_Q,  Temp_Xn_Deci4_Q,
                                Temp_Bn_Deci4_Q;
                        }

_asm mov Temp_Deci4, 1;    //Retourne du format Q2.28 (apres 2 multiplications)
_asm shl Temp_Yn_Fin_Deci4_Q, Temp_Deci4, Temp_YnTemp_Deci4_Q;
                                //au format Q1.14
_asm mov Temp_Deci4, 16;
_asm shr Temp_Yn_Fin_Deci4_Q, Temp_Deci4, Temp_Yn_Fin_Deci4_Q;
_asm nop;

//*************************************************************************
for(j=0; j<Ordre_FIR_Deci4; j++)     //Boucle Int_3
                                     //Decalage temporel des entrees Xn
        {
Xn_Deci4_Q[Ordre_FIR_Deci4-j] = Xn_Deci4_Q[Ordre_FIR_Deci4-1-j];
                        }
                }
```

```
//Decimation par 2*****************************************************

if (i % 20 == 0)                                    //IF_1
                                                    //Decimation par 4*5
        {

                Xn_Deci2_Q[0] = Temp_Yn_Fin_Deci4_Q;
                for (j=0; j<Ordre_FIR_Deci2; j++)   //Boucle Int_4
                                                    //Calcul des sorties du filtre
                        {
                                Temp_Xn_Deci2_Q = Xn_Deci2_Q[j];
                                Temp_Bn_Deci2_Q = Bn_Deci2_Q[j];
                                _asm mac.ll Temp_YnTemp_Deci2_Q, Temp_Xn_Deci2_Q,
                                Temp_Bn_Deci2_Q;
                        }

                _asm mov Temp_Deci2, 1;
                //Retourne du format Q2.28 (apres 2 multiplications)
                _asm shl Temp_Yn_Fin_Deci2_Q, Temp_Deci2, Temp_YnTemp_Deci2_Q;
                //au format Q1.14
                _asm mov Temp_Deci2, 16;
                _asm shr Temp_Yn_Fin_Deci2_Q, Temp_Deci2, Temp_Yn_Fin_Deci2_Q;
                _asm nop;

//*********************************************************************
                for(j=0; j<Ordre_FIR_Deci2; j++)    //Boucle Int_5
                                                    //Decalage temporel des entrees Xn
                        {
                                Xn_Deci2_Q[Ordre_FIR_Deci2-j]                       =
                                Xn_Deci2_Q[Ordre_FIR_Deci2-1-j];
                        }
        }

//Sortie finale du signal

                        if (i % 40 == 0)                            //IF_2
                                                                    //Decimation par 4*5*2
                        {
                                p = i/40;


// Sortie finale des decimateurs en format Q1.14
Yn_Fin_Q[p] = Temp_Yn_Fin_Deci2_Q;
```

```
                }
        }
}

//*************************************************************************
//Fin du compteur de cycles après la boucle du filtre
//*************************************************************************

_asm rdtsc timerStop;
_asm sub timerDiff, timerStart, timerStop;
_asm nop;

_asm nop;
return 0;
}
```

ANNEXE III

PLAN DE L'AMPLIFICATEUR



LISTE DE RÉFÉRENCES BIBLIOGRAPHIQUES

[1] Anritsu. 2000. « Intermodulation Distortion (IMD) Measurements: Using the 37300 Series Vector Network Analyser Application note ». En ligne. 12 p. <http:// downloadfile.anritsu.com/ RefFiles/ en-US/ Services-Support/ Downloads/ Application-Notes/ Application-Note/ 11410-00257a.pdf >. Consulté le 12 janvier 2012.

[2] Ascia, G., V. Catania, et al. (2001). "An instruction-level power analysis model with data dependency." VLSI Design 12(2): 245-273.

[3] Cai, J., P. Jiang, et al. (2010). Through-Silicon Via (TSV) capacitance modeling for 3D NoC energy consumption estimation. 2010 10th IEEE International Conference on Solid-State and Integrated Circuit Technology, November 1, 2010 - November 4, 2010, Shanghai, China, IEEE Computer Society.

[4] Callaway, T. K. et E. E. Swartzlander, Jr. (1993). Estimating the power consumption of CMOS adders. Proceedings of IEEE 11th Symposium on Computer Arithmetic, 29 June-2 July 1993, Los Alamitos, CA, USA, IEEE Comput. Soc. Press.

[5] Chen, R. Y., M. J. Irwin, et al. (1998). Architectural level hierarchical power estimation of control units. Proceedings Eleventh Annual IEEE International ASIC Conference, 13-16 Sept. 1998, New York, NY, USA, IEEE.

[6] Conte, T. M., K. N. Menezes, et al. (2000). "System-level power consumption modeling and tradeoff analysis techniques for superscalar processor design." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 8(2): 129-137.

[7] Gonzalez, G., E. Juarez, et al. (2011). Energy consumption estimation of an OMAP-based Android operating system. VLSI Circuits and Systems V, 18-20 April 2011, USA, SPIE - The International Society for Optical Engineering.

[8] Guitton-Ouhamou, P., C. Belleudy, et al. (2002). Power consumption model for the DSP OAK processor. SOC Design Methodologies. IFIP TC10/WG10.5 Eleventh International Conference on Very Large Scale Integration of Systems-on-Chip (VLSI-SOC'01), 3-5 Dec. 2001, Dordrecht, Netherlands, Kluwer Academic Publishers.

[9] Hernandez, A., L. Gomez, et al. (1993). An empirical model to estimate power consumption in GaAs DCFL/SDCFL circuits. Euromicro '92 - Short Notes, 14-17 Sept. 1992, Netherlands.

[10] Hsiao, M. S., E. M. Rudnick, et al. (2000). "Peak power estimation of VLSI circuits: new peak power measures." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 8(4): 435-439.

[11] Ishihara, T. et H. Yasuura (1996). Basic experimentation on accuracy of power estimation for CMOS VLSI circuits. Proceedings of the 1996 International Symposium on Low Power Electronics and Design, August 12, 1996 - August 14, 1996, Monterey, CA, USA, IEEE.

[12] Ishihara, T. et H. Yasuura (1997). "Experimental analysis of power estimation models of CMOS VLSI circuits." IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences E80-A(3): 480-486.

[13] Jevtic, R. et C. Carreras (2010). "Power estimation of embedded multiplier blocks in FPGAs." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 18(5): 835-839.

[14] Jevtic, R. et C. Carreras (2011). "A complete dynamic power estimation model for data-paths in FPGA DSP designs."

[15] Kang, K., J. Kim, et al. (2007). Software power estimation using IPI(inter-prefetch interval) power model for advanced off-the-shelf processor. 17th Great Lakes Symposium on VLSI, GLSVLSI'07, March 11, 2007 - March 13, 2007, Stresa-Lago Maggiore, Italy, Association for Computing Machinery.

[16] Katkoori, S. et R. Vemuri (1998). "Architectural power estimation based on behavior level profiling." VLSI Design 7(3): 255-270.

[17] Mike Tien-Chien, L., V. Tiwari, et al. (1997). Power analysis and minimization techniques for embedded DSP software. Eighth IEEE International Symposium on System Synthesis, 13-15 Sept. 1995, USA, IEEE.

[18] Nguyen, H. T., A. Chatterjee, et al. (1998). "Activity measures for fast relative power estimation in numerical transformation for low power DSP synthesis." Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology 18(1): 25-38.

[19] Rethinagiri, S. K., R. Ben Atitallah, et al. (2011). Hybrid system level power consumption estimation for FPGA-based MPSoC. 29th IEEE International Conference on Computer Design 2011, ICCD 2011, November 9, 2011 - November 12, 2011, Amherst, MA, United states, Institute of Electrical and Electronics Engineers Inc.

[20] Rita Yu, C., M. J. Irwin, et al. (2001). "Architecture-level power estimation and design experiments." ACM Transactions on Design Automation of Electronic Systems 6(1): 50-66.

[21] Russell, J. T. et M. F. Jacome (1998). Software power estimation and optimization for high performance, 32-bit embedded processors.
Proceedings of the 1998 IEEE International Conference on Computer Design, October 5, 1998 - October 7, 1998, Austin, TX, USA, IEEE.

[22] Shin, Y. et J. Lee (2006). "Power analysis of vlsi interconnect with rlc tree models and model reduction." Journal of Circuits, Systems and Computers 15(3): 399-408.

[23] Sleiman, S. B., A. Akour, et al. (2010). Millimeter-wave BiST and BiSC using a high-definition sub-ranged detector in 90nm CMOS. 53rd IEEE International Midwest Symposium on Circuits and Systems, MWSCAS 2010, August 1, 2010 - August 4, 2010, Seattle, WA, United states, Institute of Electrical and Electronics Engineers Inc.

[24] Theodoridis, G., S. Theoharis, et al. (2001). "A fast and accurate method of power estimation for logic level networks." VLSI Design 12(2): 205-219.

[25] Tiwari, V., S. Malik, et al. (1995). Instruction level power analysis and optimization of software. Proceedings of 9th International Conference on VLSI Design, 3-6 Jan. 1996, Los Alamitos, CA, USA, IEEE Comput. Soc. Press.

[26] Toner, M. F. et G. W. Roberts (1994). BIST technique for a frequency response and intermodulation distortion test of a sigma-delta ADC. Proceedings of the 12th IEEE VLSI Test Symposium, April 25, 1994 - April 28, 1994, Cherry Hill, NJ, USA, IEEE.

[27] Toner, M. F. et G. W. Roberts (1996). "Frequency response, harmonic distortion, and intermodulation distortion test for BIST of a sigma-delta ADC." IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 43(8): 608-613.

[28] Tran, D., K. K. Kim, et al. (2005). Power estimation in digital CMOS VLSI chips. IMTC'05 - Proceedings of the IEEE Instrumentation and Measurement Technology Conference, May 16, 2005 - May 19, 2005, Ottawa, ON, Canada, Institute of Electrical and Electronics Engineers Inc.

[29] Tremblay, J.-P. (2009). Analyse de performance multi-niveau et partitionnement d'application radio sur une plateforme multiprocesseur. Mémoire de maîtrise en génie électrique, Montréal, École Polytechnique de Montréal.

[30] Wang, C.-Y. et K. Roy (1996). Maximum power estimation for CMOS circuits using deterministic and statistic approaches. Proceedings of the 1996 9th International Conference on VLSI Design, January 3, 1996 - January 6, 1996, Bangalore, India, IEEE.

[31] Wang, C.-Y. et K. Roy (1998). "Maximum power estimation for CMOS circuits using deterministic and statistical approaches." IEEE Transactions on Very Large Scale Integration (VLSI) Systems 6(1): 134-140.

[32] Weste, N. H. E. et D. Harris (2005). CMOS VLSI Design: A Circuit and System Perspective. Boston, Pearson Education Inc.

[33] Bainbridge, W. J. and S. B. Furber (2001). Delay insensitive system-on-chip interconnect using 1-of-4 data encoding. Asynchronous Circuits and Systems, 2001. ASYNC 2001. Seventh International Symposium on.

[34] Furber, S. B. and P. Day (1996). "Four-phase micropipeline latch control circuits." Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 4(2): 247-253.

[35] Furber, S. B., J. D. Garside, et al. (1999). "AMULET2e: an asynchronous embedded controller." Proceedings of the IEEE 87(2): 243-256.

[36] Martin, A. J., A. Lines, et al. (1997). The design of an asynchronous MIPS R3000 microprocessor. Advanced Research in VLSI, 1997. Proceedings., Seventeenth Conference on.

[37] Stevens, K. S., S. Rotem, et al. (2001). "An asynchronous instruction length decoder." Solid-State Circuits, IEEE Journal of 36(2): 217-228.

[38] Woods, J. V., P. Day, et al. (1997). "AMULET1: an asynchronous ARM microprocessor." Computers, IEEE Transactions on 46(4): 385-398.

[39] Sutherland, I. E. (1989). "Micropipelines, The 1988 Turing Award Lecture." Comm. ACM, vol 32, no 6, June 1989, 720-738.